Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Estudio preliminar del diseño de una nueva unidad de control del motor para el vehículo FOX

Autor: Pablo Marín Cortés

Tutor: Francisco Gavilán Jiménez

Departamento de Ingeniería Aeroespacial y Mecánica de Fluidos
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Estudio preliminar del diseño de una nueva unidad de control del motor para el vehículo FOX

Autor:
Pablo Marín Cortés

Tutor:
Francisco Gavilán Jiménez

Departamento de Ingeniería Aeroespacial y Mecánica de Fluidos
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado: Estudio preliminar del diseño de una nueva unidad de control del motor para el vehículo FOX

Autor: Pablo Marín Cortés
Tutor: Francisco Gavilán Jiménez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

En primer lugar me gustaría agradecer a mis padres, mi hermana y demás familiares por el apoyo y la comprensión que me han prestado a lo largo de la realización del grado.

Hay muchas personas que me han acompañado a lo largo de estos últimos años a los que estaré eternamente agradecido, y quién sabe si habría logrado llegar a este punto de mi vida sin alguno de ellos siquiera. Estas personas me han aguantado en los peores momentos (que no es poco) y merecen por ello mi respeto y admiración. A todos ellos: gracias.

En cuanto a la realización de este proyecto, me gustaría agradecer a todos los miembros del FCCL por su inestimable ayuda, en especial a Juan José Márquez por el tiempo que le he robado en el laboratorio. Mis agradecimientos también para Carlos Bordons y mi tutor Francisco Gavilán, sin los que no habría sido posible realizar este proyecto.

Resumen

Este Trabajo de Fin de Grado se encuentra segmentado en tres grandes bloques:

De un lado se introduce y documenta la información relevante en cuanto al sistema de control electrónico del FOX, un vehículo eléctrico híbrido desarrollado por el Laboratorio de Control de Células de Combustible de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. Mediante esta información se pretende introducir en el ámbito del proyecto a un nuevo colaborador en poco tiempo.

Acto seguido, se analizan diversas propuestas de mejora del mismo a nivel de *hardware* y *software*, explicándose las principales ventajas e inconvenientes de cada una y valorándose los requisitos necesarios del vehículo para llevarlas a cabo.

Por último se va un paso más allá estudiando un Sistema Operativo para la Unidad de Control Electrónica basado en *Linux*, para lo cual se introducen conceptos clave como *tiempo real*, *kernel*... y se explica la manera práctica de implementarlo desde los SO instalados en la mayor parte de los ordenadores personales actuales, de manera que la prueba se pueda recrear en el laboratorio o servir como base para estudios y ensayos más avanzados.

El objeto del proyecto combina pues algunas pinceladas de diseño de sistemas de control electrónicos embebidos con un estudio de las características de dispositivos comerciales y sistemas operativos que cumplan los requisitos impuestos por un entorno de control de gran exigencia. No obstante debe notarse el hecho de que el nivel de profundidad con el que se estudian los distintos conceptos está enfocado a la comprensión a nivel práctico más que a un estudio teórico exhaustivo.

Abstract

This project can be divided in three blocks:

First, some relevant information on the electronic control unit (ECU) of the FOX vehicle is introduced, having been the latter developed by the *Fuel Cell Control Lab* Group from the Escuela Técnica Superior de Ingeniería of the University of Sevilla. This is so a new partner can be introduced to the enviroment of the project as soon as possible.

Then several updates are analized software and hardware-wise, explaining the main pros and cons of each one and evaluating the vehicle's requirements to implement them.

Lastly, a further step is taken by studying an operating system for the electronic control unit based on *Linux*, some key concepts are introduced, such as *real time*, *kernel*... and the practical way of implemeting it from the most commonly available OS on personal computers is described, so that the test can be recreated on the lab or to serve as a basis for more advanced ones.

Therefore, the aim of the project combines some sketches of embedded electronic control units design with an analysis of comercial devices'features and operating systems which comply with the hard control requirements of such a complex enviroment. Nevertheles, it may be noted that the level of deepness by which most of the terms are described is intended for a practical understanding rather than for an exhaustive theoretical explanation.

Índice

1 Introducción

1.1 Reseña histórica: La electrónica embarcada.

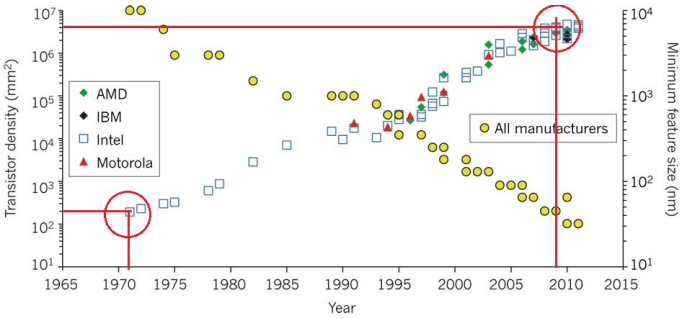
Actualmente , la electrónica y su aplicación en la computación juegan un papel fundamental en el control automático de la mayoría de sistemas relevantes en la ingeniería. La motivación detrás de esta realidad se encuentra en la creciente complejidad de los sistemas y sus variables, que desbancan a soluciones tradicionales (i.e. control por medios mecánicos) por imposibles o poco eficientes para los criterios actuales. A esta situación se ha llegado a través de una evolución relativamente rápida, marcada por los grandes descubrimientos del siglo pasado en materia de la electrónica, entendida como la ciencia que estudia las magnitudes eléctricas como medio para tratar la información, como los tubos de vacío (Edison y Lee de Forest en la primera década) y los semiconductores (Shockley, Bardeen y Brattain a finales de los años cuarenta y durante los cincuenta). Tales avances sentaron las bases para la revolución tecnológica actual.

En el contexto del presente texto, durante la primera mitad del s.XX la industria automotriz llegó a presentar un cierto grado de madurez tecnológica. No obstante, los automóviles de esta época, por norma general, no incluían en modo alguno sistemas electrónicos, quedando todas las leyes de control (inyección de combustible, encendido de bujías,... etc) implementadas de manera mecánica. La tendencia comenzó a revertirse en la década de los 70 con una crisis petrolífera que llevó a la regulación federal en E.E.U.U. sobre las emisiones de los vehículos, y que sentó las bases para que la industria del automóvil se replanteara los métodos de control obsoletos en busca de un grado de sostenibilidad mayor.

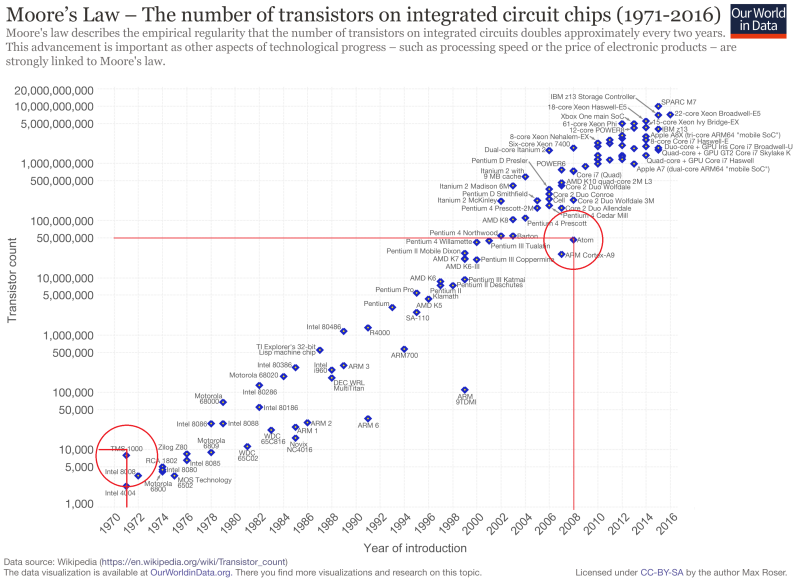
El escollo más importante con el que se tuvo que lidiar en los primeros modelos fue el tamaño de la electrónica de la época, pues ésta se encontraba en los primeros pasos de su desarrollo. Tal problemática se encuentra en la actualidad resuelta en términos satisfactorios, si bien la miniaturización de la electrónica continúa aun incluso a día de hoy.

A modo de ilustración, tal y como se puede ver en la figura 1.1a ([2]), a comienzos de la década de los 70 la técnica industrial de fabricación de semiconductores era capaz de producir circuitos integrados con una densidad del orden de 200 transistores/mm^2. La ley de Moore, representada en la figura 1.1b ([3]), evalúa la tendencia del número de transistores en un circuito integrado con el tiempo, e indica un número de 10000 transistores para el modelo típico de la época *TMS-1000* (microprocesador fabricado por *Texas Instruments* en 1970). Una estimación burda de la superficie de tal circuito integrado podría ser entonces 10000 transistores$/200$ transistores$/mm^2 = 50mm^2$, siendo su frecuencia máxima de trabajo $0.4\ MHz$, mientras que un microprocesador moderno de la serie *Intel Atom* puede llegar a los $1.6\ GHz$ con 50 millones de transistores según sus características nominales, ocupando un área de tan solo 5×10^7 transistores$/10^7$ transistores$/mm^2 = 5mm^2$ en virtud de los datos señalados en los gráficos de la figura 1.1 . Estos simples cálculos dejan claro que el tamaño y el peso necesarios para implantar un sistema de control electrónico en un vehículo se han visto astronómicamente reducidos, lo cual abre la puerta a una gran diversidad de soluciones de control embebidas entre las que se encuentran las consideradas en este proyecto.

A todo lo comentado hay que sumar el hecho de que el número de variables a controlar se ha visto incrementado notablemente; frente al único control electrónico de la inyección en la década de los 70, un vehículo actual puede disponer de centenares de microcontroladores encargados de tareas que varían desde el elevalunas eléctrico y la gestión de puertas abiertas hasta las más complejas como el control de tracción y las emisiones.



(a) Densidad en transistores/mm^2 y tamaño de puerta en *nm*. Los valores correspondientes a los microprocesadores de referencia se encuentran indicados en color rojo .



(b) Ley de Moore: número de transistores de un circuito integrado .

Figura 1.1 Tamaño de los circuitos integrados en los últimos 50 años.

Algunos hitos de la computación en el automóvil

- Históricamente, el primer modelo comercial en implementar un sistema electrónico de inyección fue el *Tipo 3* de *Volkswagen* en 1968.
- Un año más tarde, *Ford* comenzó a usar un sistema electrónico de anti-skid.
- En 1971 *General Motors* introduce una unidad electrónica de control de tracción.
- A mediados de los años 80 surge el concepto de OBD (*On Board Diagnostics*), que pasó a ser obligatorio en todos los vehículos a partir de 1991 con el objetivo de reducir las emisiones. En 1996 se avanzó en su definición mediante la introducción del estándar OBD II.

1.2 Vehículos Eléctricos Híbridos

Los vehículos con motor de combustión interna (ICE por sus siglas en inglés *Internal Combustion Engine*) han conformado la mayoría de la oferta de mercado desde los orígenes de la automoción. Sin embargo, los vehículos eléctricos de batería ofrecieron batalla hasta la bajada del precio del petróleo tras la crisis mencionada en la sección anterior (década de los 70) y hasta que la capacidad de carga de sus baterías se convirtió en el cuello de botella de esta tecnología, acabando prácticamente con su salida del mercado.

Motivada por la búsqueda de la eficiencia y del escaso impacto ambiental, tal tendencia se encuentra actualmente en retroceso en todos los sectores del transporte. En el caso de la industria aeronáutica, predomina la filosofía *More Electric Aircraft* que plantea que las fuentes de energía de una aeronave evolucionen progresivamente hasta ser completamente eléctricas.

En los términos cubiertos por este trabajo, los vehículos híbridos que se consideran son los de tipo FCHV o *Fuel Cell Hybrid Vehicles*, entendiendo como tales a los que utilizan dos fuentes de energía electrica; La almacenada en baterías convencionales (como las de Li-Ion) se combina con la generada en una pila de combustible, un dispositivo electroquímico que extrae energía eléctrica a partir de la energía química almacenada en un combustible.

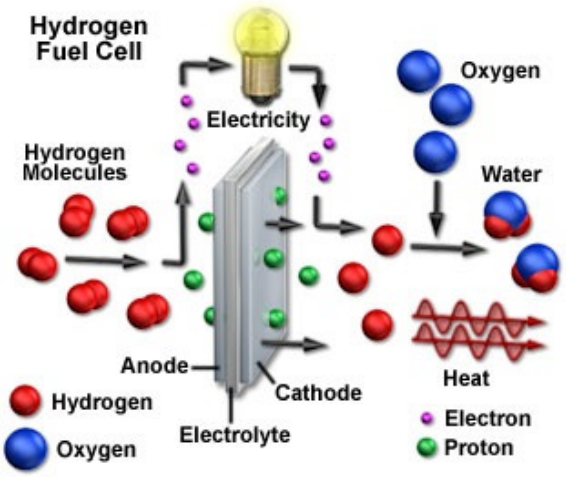


Figura 1.2 Esquema de una pila de hidrógeno.

1.3 El vehículo FOX

El vehículo eléctrico híbrido FOX (figura 1.3a, ([4])) es un proyecto del Laboratorio de Control de Células de Combustible (FCCL, *Fuel Cell Control Lab*) del departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. Se trata de un chasis comercial de competición modificado al que se se añadieron cuatro motores eléctricos que lo dotan de tracción independiente para cada rueda. Con objeto de conseguir aprovechar eficientemente el par entregado, se dispone de una unidad de control electrónico que actúa sobre aspectos como la tracción y la gestión de la batería.

(a) Vehículo FOX .

(b) Laboratorio del FCCL.

Figura 1.3 Introducción al vehículo considerado.

Algunos proyectos predecesores del FOX fueron los vehículos Delfín (figura 1.4a) y Hércules (figura 1.4b)

([4]), siendo este último la adaptación de un chasis de todoterreno convencional al concepto de vehículo híbrido, y suponiendo un gran desafío en cuanto a integración de distintas disciplinas.

(a) Proyecto delfín . **(b)** Proyecto Hércules.

Figura 1.4 Proyectos predecesores al FOX.

1.4 Objetivos del bloque

El presente bloque del proyecto pretende formular la descripción del sistema de control electrónico embebido en un vehículo eléctrico híbrido concreto (El FOX, introducido en la anterior sección), para posteriormente evaluar la posibilidad de mejorarlo. La línea de pensamiento en la que está basado queda planteada en el siguiente esquema:

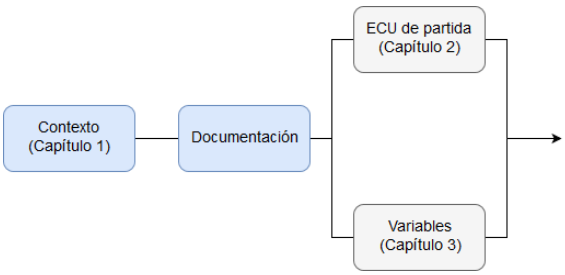


Figura 1.5 Esquema del primer bloque.

Contexto (Capítulo 1) El proyecto surge para dotar al FOX de un sistema embarcado que mejore las prestaciones del actual, consistente en varios módulos PC/104 descritos en el capítulo 2, y que permita incluir el control de nuevas variables.

Documentación (Capítulos 2 y 3) Se recopila la información más relevante procedente de los archivos del laboratorio, y se documentan los aspectos relativos al *hardware* y *software* del vehículo.

Capítulo 2. Configuración de partida Se analiza el diseño de ECU implementado, describiendo con detalle tanto su posición dentro del vehículo como sus dimensiones físicas, indicándose asimismo componentes comerciales y sus características. También se ilustran las conexiones entre los elementos que la componen y los SO que gestionan el conjunto.

Capítulo 3. Variables Se estudia de forma descriptiva el sistema de sensores y las variables asociadas a cada magnitud medida, indicando el nombre asociado a cada una en el código principal de la ECU.

2 Configuración de partida

El sistema de control electrónico del vehículo FOX se encuentra dividido en tres subsistemas que se describen en la presente sección:

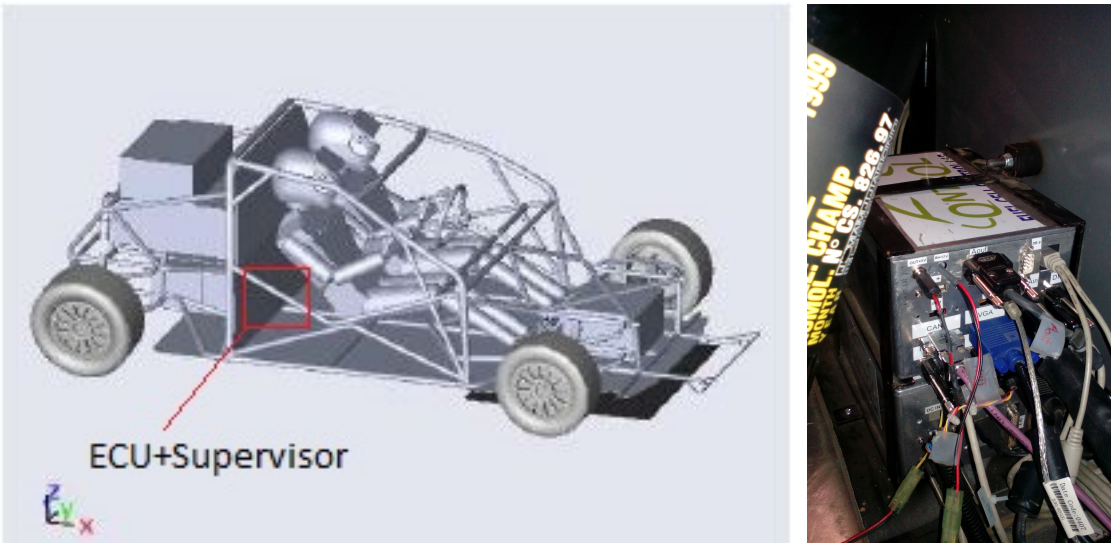
- *Electronic Control Unit* (ECU).
- Supervisor.
- *Battery Management System* (BMS).

2.1 Localización de los sistemas de control

La ECU y el supervisor se encuentran situados transversalmente tras el asiento del copiloto del vehículo FOX (fig.2.1a), cada uno aislado en un chasis de aluminio individual y refrigerado mediante un ventilador de ordenador de reducidas dimensiones. El panel de conexiones se encuentra orientado hacia el asiento del copiloto (fig.2.1b).

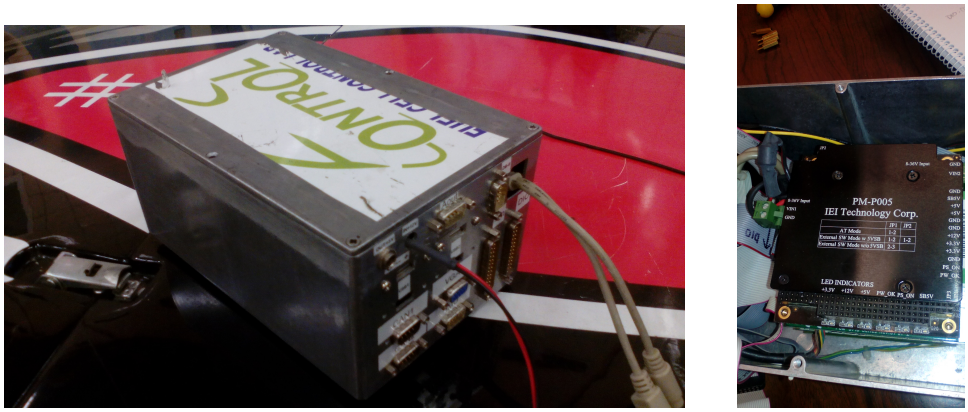
2.2 *Electronic Control Unit* (ECU)

Se trata del elemento central de control del vehículo. Consiste esencialmente en un SBC (*Single Board Computer*), esto es, un computador construido en una única placa o PCB, que incluye los elementos funcionales básicos del mismo: microprocesador, memoria y entradas/salidas.



(a) Localización del bloque ECU+Supervisor en el FOX. **(b)** Panel de conexiones.

Figura 2.1 Posición de la ECU.



(a) Vista general. **(b)** Interior.

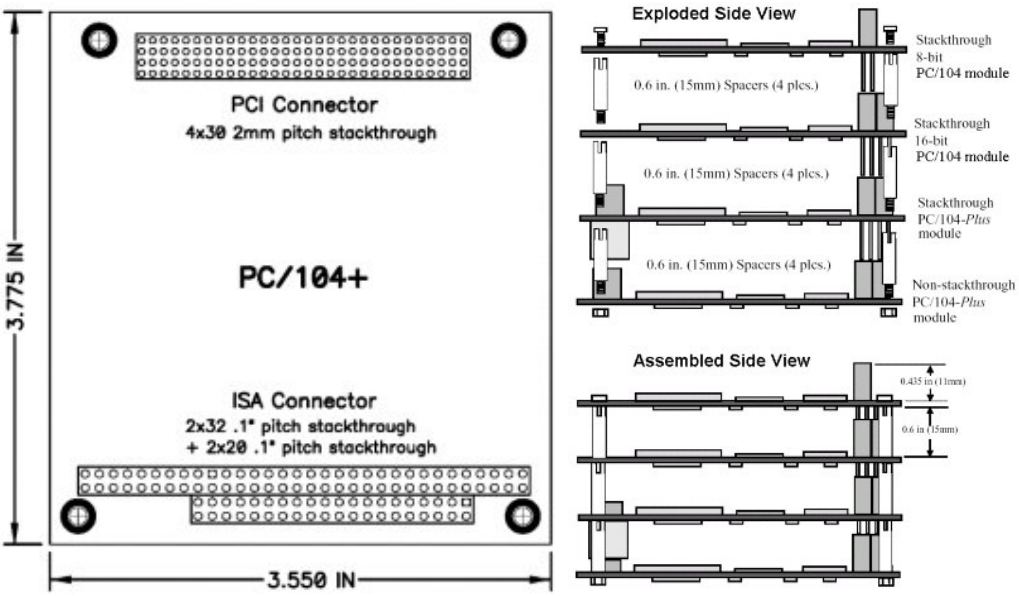
Figura 2.2 Electronic Control Unit.

2.2.1 Factor de forma PC/104

Un factor de forma consiste en una serie de estándares que definen las dimensiones físicas de la placa madre de un computador. Tal conjunto incluye tanto las propias dimensiones de la placa del circuito donde se encuentran los componentes (RAM, CPU,... etc) como la fuente de alimentación, puertos de comunicación y aspectos referentes al montaje en el chasis del ordenador como agujeros de fijación (número, espaciado,... etc).

El factor de forma PC/104 fue ideado por *Ampro* en 1987, aunque no fue estandarizado hasta el año 1992 por la institución IEEE. Las dimensiones generales son 3.6×3.8 pulgadas (90×96 mm.), tal y como se recoge en la figura 2.3a

El estándar de PC/104 usado en la ECU del FOX es el PC/104+ (plus). Este difiere del PC/104 en que incluye soporte para puerto PCI (*Peripheral Component Interconnect*) además de para ISA (*Industry Standard Architecture*), incluido también en el primer formato.



(a) Vista en planta del factor de forma PC/104. **(b)** *Stack* de tarjetas PC/104 .

Figura 2.3 Factor de forma PC/104 ([5]).

Una de las características distintivas del factor de forma es que los módulos de expansión del SBC se apilan encima del mismo, reduciendo el espacio físico total ocupado por el computador en favor de la compacidad

(fig. 2.3b). Otra de sus ventajas son el bajo consumo energético y un diseño pensado para trabajar en entornos con vibraciones y altas temperaturas, las cuales hacen de este factor de forma una buena elección para aplicaciones embarcadas.

2.2.2 Diamond Systems PCM-3370

El corazón de la ECU está compuesto por el modelo comercial de SBC *PCM-3370*, lanzada en 2003 por la empresa californiana *Diamond Systems Corporation*. Sus características pueden encontrarse en el *Datasheet* facilitado por el fabricante. Seguidamente se incluyen y comentan algunas de las más importantes.

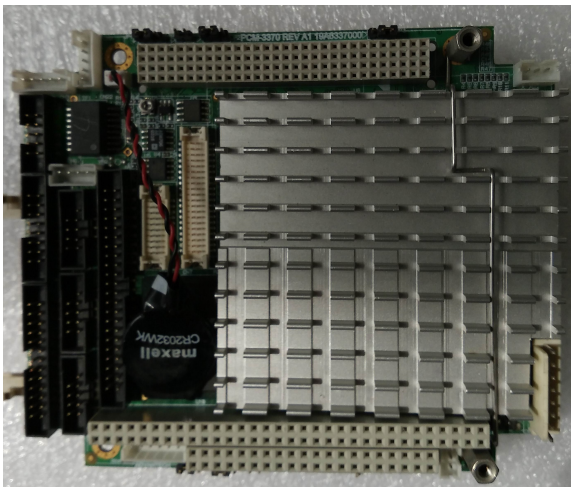


Figura 2.4 *SBC PCM-3370* ([6]).

CPU: Pentium III La Unidad Central de Procesamiento o procesador es el componente de un ordenador encargado de procesar datos, funcionando conjuntamente con la RAM.

DRAM: 1 ranura SODIMM , con soporte de hasta 512MB SDRAM. La RAM Dinámica (DRAM) es el tipo de memoria volátil de acceso aleatorio que necesita realizar ciclos de refresco para mantener los datos almacenados, al contrario que la SRAM (*Static RAM*), que mantiene la información mientras se encuentra alimentada. SDRAM (*Synchronous Dynamic RAM*) hace referencia a que el cambio de estado se encuentra sincronizado con la señal de reloj del sistema.

SSD: tarjeta *Compact Flash* de Tipo I. SDD son las siglas de *Solid State Drive*, dispositivo de almacenamiento de memoria no volátil (al contrario que la RAM) y de estado sólido (En contraposición a las unidades de disco duro mecánicas convencionales o HDD).

CAN: 1 puerto de bus CAN 2.0. *Controller Area Network* es un estándar de bus de datos con uso extensivo en los vehículos modernos. Básicamente se puede afirmar que consiste en el principal elemento de unión entre los distintos sistema que componen una unidad de control electrónico embebida. En el Anexo A se detallan los aspectos básicos del estándar.

2.2.3 PCM 3718 HO

Se trata de un módulo PC/104 fabricado por *Advantech* con entradas analógicas, además de E/S digitales:

- Entradas Analógicas

  Canales: 16 canales individuales/8 diferenciales

  Resolución: 12 bits

  Velocidad de muestreo máxima: 100KHz

  Impedancia de entrada: 10MΩ. En el ámbito de la conexión entre sistemas, para que la medida de voltaje afecte lo menos posible a la fuente del mismo, se diseña la carga (la tarjeta PC/104 en este caso) con una impedancia muy superior a la de la fuente, de manera que no se requiera demasiada corriente procedente de la misma.

  Modos de muestreo: *software, pacer*, externo

Rango de entrada: Unipolar; $0 \sim 10V$ $0 \sim 5V$ $0 \sim 2.5V$ $0 \sim 1.25V$

- Salidas Analógicas

Canales: 1 (12 bits). Esta característica es distintiva del modelo de placa PCM 3718 en su versión HO.

Rango de salida:

Referencia interna	Unipolar (V)	$0 \sim 5$, $0 \sim 10$
Referencia externa (V)	$0 \sim 10$, $0 \sim -10$	

Impedancia de salida: 0.1Ω máx. En esta etapa, al contrario que en la entrada, la impedancia debe ser lo más parecida posible a la de la carga, que típicamente es muy baja, para que la transferencia de energía sea lo mayor posible.

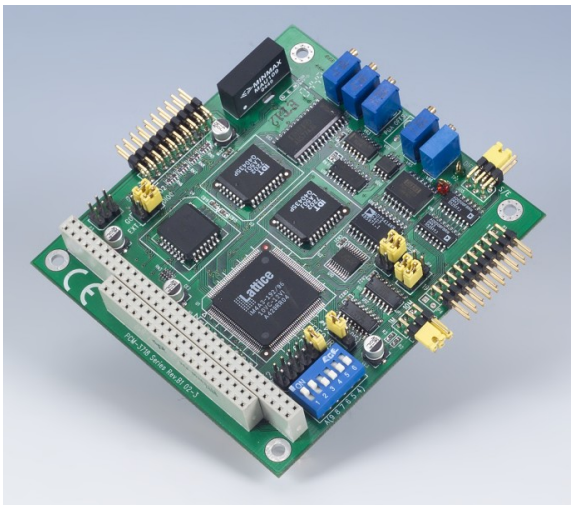


Figura 2.5 Módulo PCM 3718 HO ([7]).

2.2.4 RUBY-MM-1612 V3

Consiste en un módulo de salidas analógicas y de entradas/salidas digitales. Económicamente es el componente más costoso, debido a la mayor precisión que se consigue comparada con la de una salida digital. Entre sus principales características se encuentran:

- Salidas Analógicas

Canales: 16 canales de 12 bits

Rangos de salida: $\pm 5V, \pm 10V, 0-5V, 0-10V$

- I/O Digitales

Canales: 24

Rangos lógicos: Esta especificación indica los valores límite de voltaje para considerar 0 o 1 lógico dependiendo de si la línea es de entrada o de salida:

	Voltaje de entrada	Voltaje de salida
0 lógico	-0.5V min., 0.8V máx.	0.0V min, 0.4V máx.
1 lógico	2.0V min., 5.5V máx.	3.0V min., 4.6V máx.

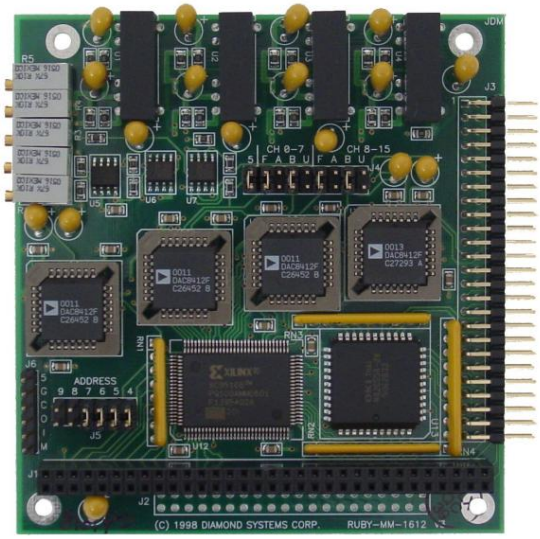


Figura 2.6 RUBY-MM-1612 V3 ([8]).

2.2.5 PM-P005

Constituye la fuente de alimentación de la ECU. Está fabricada por *IEI Technology Group*. Cabe destacar que está diseñada para funcionar entre 8 y 36 VDC, lo cual hace posible afrontar cualquier caída de tensión eventual sin comprometer al control del vehículo.

Figura 2.7 PM-P005 ([9]).

2.2.6 Distribución modular

Las tarjetas descritas en la sección anterior se disponen en el interior de la ECU apiladas en sentido vertical según se muestra en la figura 2.3b. Tal y como se puede apreciar en la 2.2b, de arriba a abajo se encuentran la fuente de alimentación, las tarjetas de salidas y entradas analógicas *Ruby-MM-1612 V3* y *PCM-3718 HO*, la principal *PCM 3370* y, por último la tarjeta controladora del bus CAN. Se puede visualizar el montaje en la figura 2.8

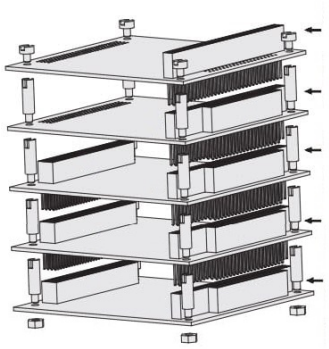


Figura 2.8 Esquema conjunto de la ECU.

2.2.7 Sistema Operativo

El sistema operativo (OS) con el que se administra la unidad electrónica de control es *QNX 6.3.2*.

Los orígenes de este sistema, según la propia página web de la compañía (actualmente subsidiaria de *Blackberry*), se remontan a 1980.

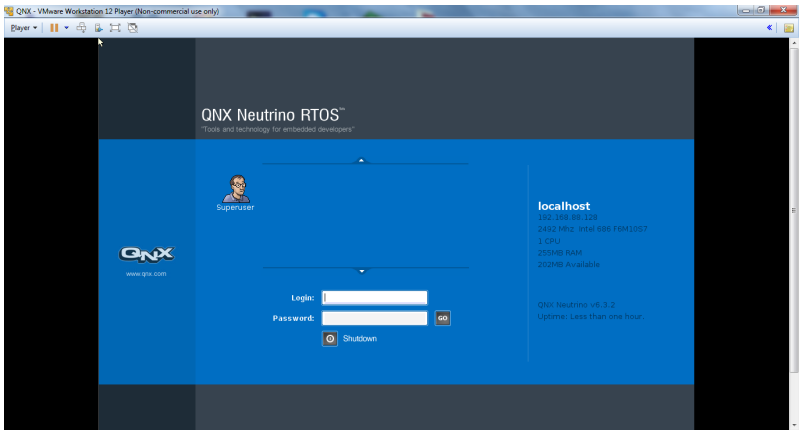


Figura 2.9 Ventana de inicio de sesión en *QNX*.

La versión implantada en el FOX es la 6.3.2, lanzada en 2007. Este SO ha desarrollado su operación satisfactoriamente en los últimos años e incluso hoy en día es utilizado para realizar las pruebas del vehículo. No obstante presenta una serie de problemas, tal y como constata el personal de laboratorio del FCCL, entre los que se encuentran el bloqueo completo y aparentemente aleatorio del sistema al utilizar los medios extraíbles necesarios para transferir los controladores, la reducida compatibilidad con *drivers* más modernos y el cese del soporte del SO por parte de la compañía. Todas estas consideraciones llevan a la conclusión de que se dispone de un sistema obsoleto para los estándares actuales implantado en el vehículo que presenta un gran potencial de mejora.

Otro aspecto a tener en cuenta es que la mayoría de los controladores del vehículo (tracción, estabilidad, hilos de comunicación entre sistemas...) se encuentran programados en lenguaje QNX, por lo que para cualquier SO distinto sería necesario adaptar sus correspondientes códigos. Esta eventualidad debería ser una línea futura de acción en el proyecto FOX y se comentará posteriormente.

2.3 Supervisor

Está situado en otro chasis metálico bajo la ECU y se encarga de la gestión de la seguridad en su operación, haciéndose cargo de aspectos tales como paradas de emergencia y protección eléctrica. El sistema operativo que emplea es *Windows*. En primera instancia no se considera su actualización.

2.4 BMS

La BMS (*Battery Management System*) es un sistema con el objeto de gestionar la batería. En el caso del vehículo FOX, viene incorporada en la batería instalada. Algunas de las funciones de la BMS se recogen en [10], y son:

Protección de las celdas La aplicación fundamental de una BMS es proteger a la batería de las condiciones fuera de diseño. En la práctica la BMS debe proporcionar una protección completa de las celdas que componen la batería frente a cualquier eventualidad. Si la batería opera fuera de sus límites de diseño puede fallar catastróficamente, y esto es de particular importancia en aquellas destinadas a la automoción, donde las condiciones son hostiles y donde se encuentras sometidas al abuso del usuario.

Control de carga Es una función esencial de la BMS. Las malas condiciones de carga son la principal causa de daño en las baterías.

Gestión de demanda Aunque no se encuentra directamente relacionada con la operación de la batería en sí misma, la gestión de demanda se refiere a la aplicación para la cual se usa la batería. Su objetivo es minimizar la demanda de corriente, prolongando así el tiempo entre cargas.

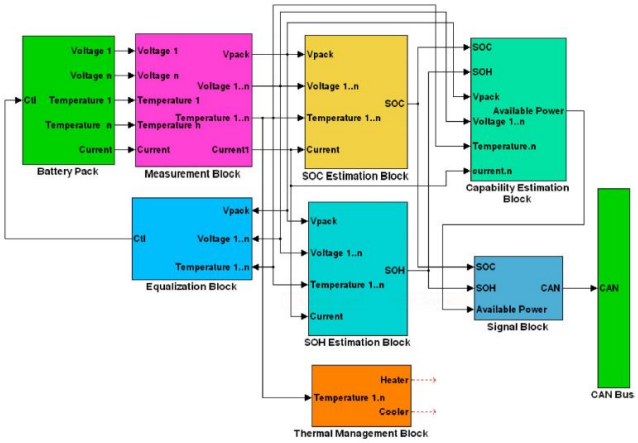
Determinación del SOC Muchas aplicaciones requieren un conocimiento del estado de carga (*State of Charge*) de la batería o de las celdas individuales que la componen. Esto puede tener como objetivo informar al usuario de la capacidad que queda en la batería, o necesitarse en un circuito de control para asegurar un proceso de carga correcto.

Determinación del SOH El estado de salud de la batería (*State Of Health*) es una medida de la capacidad de la misma para proporcionar su salida nominal. Esto es vital para indicar la disponibilidad de los equipos de energía de emergencia y es un indicador de servicio de la batería. *Equilibrado de celdas* En baterías compuestas por una cadena de celdas, las pequeñas diferencias en las tolerancias de producción o de condiciones de operación tienden a pronunciarse con cada ciclo de carga y descarga. Las celdas más débiles se encuentran sobrecargadas durante la carga, volviéndose incluso más débiles, hasta que al final fallan causando la destrucción prematura de la batería. El equilibrado de celdas es una manera de compensar estas diferencias, garantizando que la carga de todas ellas será igual y por tanto extendiendo la vida de la batería.

Historial de carga Otra posible función de la BMS es la monitorización de la historia operacional de la batería. Esta característica es necesaria para estimar el SOH, además de para determinar si se ha hecho un uso abusivo de la batería. Parámetros tales como el número de ciclos, los voltajes y temperaturas máximos y mínimos y las corrientes máximas y mínimas de carga y descarga se pueden almacenar para su consecuente evaluación. Esta puede ser una herramienta importante para verificar una operación bajo las condiciones de la garantía del fabricante.

Autenticación e identificación La BMS también permite la posibilidad de almacenar información sobre la celda como la designación del fabricante y la química de la misma, lo que puede facilitar comprobaciones automáticas y el número de serie y de lote pueden facilitar el seguimiento en caso de fallo.

Comunicación la mayoría de los sistemas BMS incorporan algún tipo de comunicación entre la batería y el equipo de carga o prueba. Algunos poseen conexiones con otros sistemas que sirven de interfaz con la batería para monitorizar sus condiciones o su historial. Las interfaces de comunicación también son necesarias para permitir acceso a la batería por parte del usuario para monitorizar los parámetros de control de la BMS y para tests y diagnósticos. En el caso del FOX, se encuetra conectada a la ECU a través de bus CAN.



(a) Diagrama de bloques de la BMS.

(b) Vista lateral donde se aprecian la HMI y la pantalla de la BMS.

Figura 2.10 *Battery Management System.*

2.5 Interconexión general de sistemas

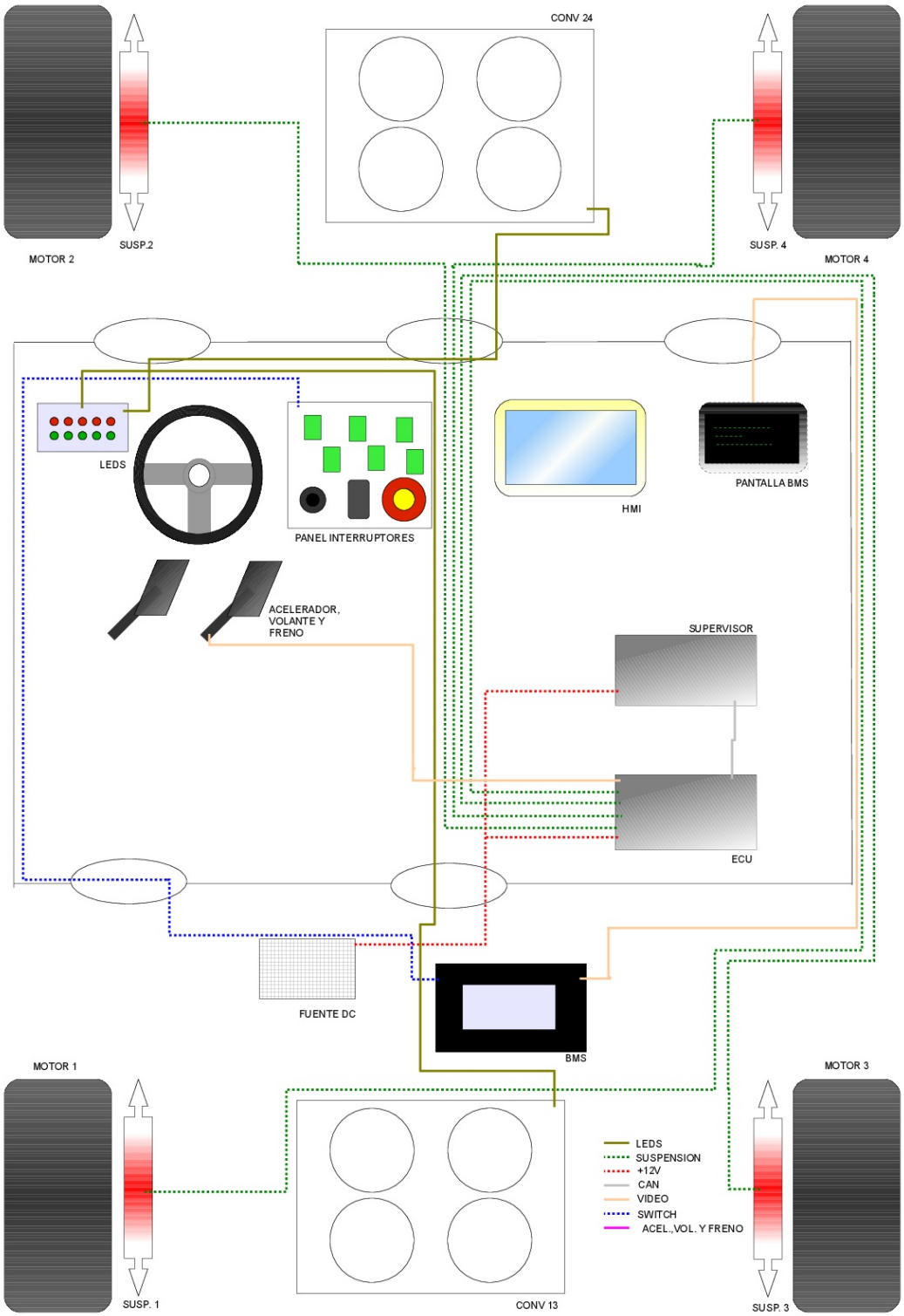


Figura 2.11 Esquema de cableado.

3 Variables

La finalidad de este capítulo es documentar las variables manejadas por la unidad de control electrónico. Esta tarea es necesaria y de vital importancia a la hora de elegir nuevo *hardware*, ya que resulta determinante para determinar la necesidad o no de expansión de un computador comercial dadas sus características en cuanto a entradas y salidas.

Los datos presentados en las tablas 3.1, 3.2 y 3.3 provienen de dos fuentes principales. Por un lado se revisaron todos los archivos del proyecto almacenados en los ordenadores del laboratorio y por otro se emuló el sistema operativo del vehículo en una *máquina virtual*, donde se consultaron los códigos de los programas principales de adquisición de datos.

3.1 Sensores

El vehículo incluye una serie de sensores con dos objetivos principales: medir la posición de los dispositivos del conductor y conocer la posición y el comportamiento del vehículo. La distribución de los mismos se encuentra detallada a continuación e ilustrada en la figura 3.1.

- Para el primero se utilizan tres potenciómetros, que miden el recorrido de los pedales de aceleración y freno, además del ángulo de volante.
- Para el segundo objetivo se utilizan cuatro potenciómetros para medir el recorrido de las suspensiones (SDI y SDD de la figura3.1), una Unidad de Medida Inercial (IMU) para medir las aceleraciones y velocidades (lineales y angulares) en los tres ejes y un GPS, embebido en ésta última, que proporciona localización geográfica y velocidad.

A esto hay que añadir los datos que recogen los sensores internos de las baterías (SOC, tensión, intensidad) y de los motores y convertidores (intensidad, velocidad angular del motor).

3.2 Lista de Variables

A continuación se presentan las principales variables recogidas, junto con la ruta a través de la que se transmiten y el rango de tensiones entre los que se encuentran.

3.2.1 Entradas Analógicas

Son las introducidas en el módulo PC/104 *PCM 3718 HO* de la ECU, cuya sección de E/S digitales es referida en la documentación del laboratorio como Tarjeta de Adquisición Digital o TAD. El conector destinado a esta aplicación situado en el panel de conexiones de la figura 3.5 es de tipo *RS-232 DB-25*, ilustrado en el esquema 3.2

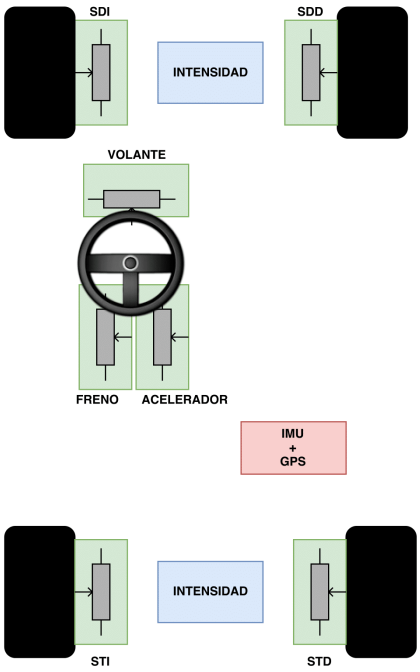


Figura 3.1 Esquema de los sensores en el FOX.

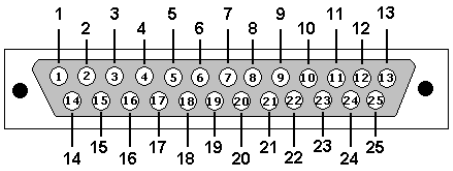


Figura 3.2 Conector *RS-232 DB-25*.

Tabla 3.1 Lista de entradas analógicas.

Pin TAD	Nº canal	Nombre canal	Uso	Rango	Conector	Pin DB-25	Origen
1	A/D S0	POS_VECTOR_ACL	Lee el acelerador del vehículo	0<->5V	DB-25 A.IN	1	Potenciómetro del acelerador
3	A/D S1	POS_VECTOR_FRN	Lee el freno del vehículo	0<->5V	DB-25 A.IN	3	Potenciómetro del freno
5	A/D S2	POS_VECTOR_VLT	Lee la posición de giro del volante	0<->5V	DB-25 A.IN	5	Potenciómetro del volante
7	A/D S3	POS_VECTOR_STI	Lee la posición de la suspensión trasera izquierda	0<->5V	DB-25 A.IN	7	Potenciómetro trasero izquierdo (STI)
9	A/D S4	POS_VECTOR_STD	Lee la posición de la suspensión trasera derecha	0<->5V	DB-25 A.IN	9	Potenciómetro trasero derecho (STD)
11	A/D S5	POS_VECTOR_SDI	Lee la posición de la suspensión delantera izquierda	0<->5V	DB-25 A.IN	11	Potenciómetro delantero izquierdo (SDI)
13	A/D S6	POS_VECTOR_SDD	Lee la posición de la suspensión delantera derecha	0<->5V	DB-25 A.IN	13	Potenciómetro delantero derecho SDD
15	A/D S7	POS_VECTOR_I_F	Lee la corriente consumida/cedida por el eje delantero	-4<->4V	DB-25 A.IN	15	Sensor de corriente delantero
2	A/D S8	POS_VECTOR_I_R	Lee la corriente consumida/cedida por el eje trasero	-4<->4V	DB-25 A.IN	2	Sensor de corriente trasero
4	A/D S9	-	-	-	DB-25 A.IN	4	-
6	A/D S10	-	-	-	DB-25 A.IN	6	-
8	A/D S11	-	-	-	DB-25 A.IN	8	-
10	A/D S12	-	-	-	DB-25 A.IN	10	-
12	A/D S13	-	-	-	DB-25 A.IN	12	-
14	A/D S14	-	-	-	DB-25 A.IN	14	-
16	A/D S15	-	-	-	DB-25 A.IN	16	-
17	A.GND	-	-	-	DB-25 A.IN	17	-
18	A.GND	-	-	-	DB-25 A.IN	18	-
19	A.GND	-	Se puede usar como la salida analógica PCM-3718-HO	-	DB-25 A.IN	19	-
20	A.GND	-	*Se puede usar como la tierra de la señal anterior	-	DB-25 A.IN	20	-
21	-	-	GND APANTALLAMIENTO CABLES	0	DB-25 A.IN	21	-
22	-	-	GND	0	DB-25 A.IN	22	-
23	-	-	NARANJA (LIBRE)		DB-25 A.IN	23	-
24	-	-	+5V VOL, ACEL, FRENO	0<->5V	DB-25 A.IN	24	-
25	-	-	+5V SENSORES DE SUSPENSIÓN	0<->5V	DB-25 A.IN	25	-

3.2.2 Salidas Analógicas

En este caso, el dispositivo encargado de producir las señales es el *Ruby-MM-1612* con conector *RS-232 DB-9*, representado en 3.3 y marcado en el panel como *Aout*.

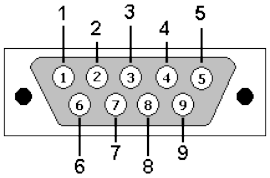


Figura 3.3 *RS-232 DB-9*.

Tabla 3.2 Lista de salidas analógicas.

Pin TAD	Nº canal	Nombre canal	Uso	Rango	Conector	Pin DB-9	Destino
2	Vout_0	ANLG_OUT_ACL_M1	Escribe señal de acelerador para M1	0<->5V	DB-9	8	Motor 1
4	Vout_1	ANLG_OUT_ACL_M2	Escribe señal de acelerador para M2	0<->5V	DB-9	7	Motor 2
6	Vout_2	ANLG_OUT_ACL_M3	Escribe señal de acelerador para M3	0<->5V	DB-9	6	Motor 3
8	Vout_3	ANLG_OUT_ACL_M4	Escribe señal de acelerador para M4	0<->5V	DB-9	5	Motor 4
10	Vout_4	ANLG_OUT_FRN_M1	Escribe señal de freno para M1	0<->5V	DB-9	4	Motor 1
12	Vout_5	ANLG_OUT_FRN_M2	Escribe señal de freno para M2	0<->5V	DB-9	3	Motor 2
14	Vout_6	ANLG_OUT_FRN_M3	Escribe señal de freno para M3	0<->5V	DB-9	2	Motor 3
16	Vout_7	ANLG_OUT_FRN_M4	Escribe señal de freno para M4	0<->5V	DB-9	1	Motor 4
15	A.GND	-	Tierra	GND	DB-9	9	-

3.2.3 E/S Digitales (D.I/O)

Proporcionadas por la tarjeta *PCM-3718*, los actuadores controlados por este tipo de señales son principalmente los interruptores de activación de frenos y aceleradores del vehículo consistentes en relés activados con los 5VDC de salida de dicho módulo. (Obsérvese la figura 3.4)

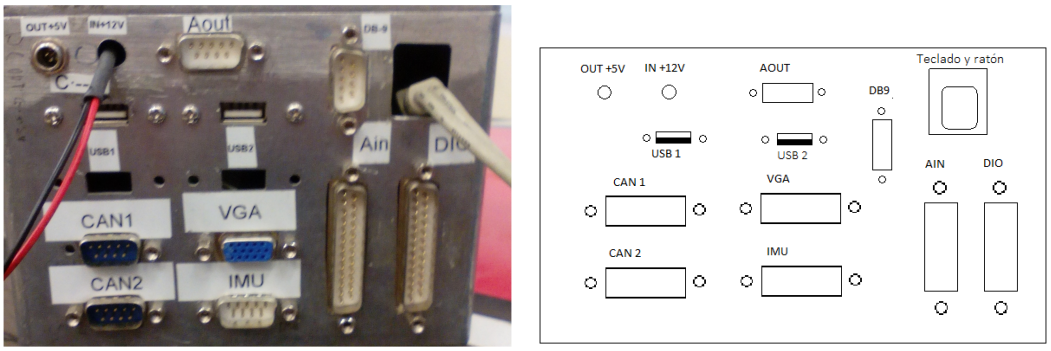
Tabla 3.3 Lista de entradas/salidas digitales.

Pin TAD	Nº canal	Nombre canal	Uso	Rango	Conector	Pin DB-25
1	DIO 0	DIG_IN_REV	El estado del interruptor de marcha atrás	0/5V	DB-25	1
2	DIO 1	DIG_OUT_ACL_M1_ON	Emula la activación del interruptor del acel. del motor 1	0/5V	DB-25	2
3	DIO 2	DIG_OUT_ACL_M2_ON	Emula la activación del interruptor del acel. del motor 2	0/5V	DB-25	3
4	DIO 3	DIG_OUT_ACL_M3_ON	Emula la activación del interruptor del acel. del motor 3	0/5V	DB-25	4
5	DIO 4	DIG_OUT_ACL_M4_ON	Emula la activación del interruptor del acel. del motor 4	0/5V	DB-25	5
6	DIO 5	DIG_OUT_FRN_M1_ON	Emula la activación del interruptor del freno del motor 1	0/5V	DB-25	6
7	DIO 6	DIG_OUT_FRN_M2_ON	Emula la activación del interruptor del freno del motor 2	0/5V	DB-25	7
8	DIO 7	DIG_OUT_FRN_M3_ON	Emula la activación del interruptor del freno del motor 3	0/5V	DB-25	8
9	DIO 8	DIG_OUT_FRN_M4_ON	Emula la activación del interruptor del freno del motor 4	0/5V	DB-25	9
10	DIO 9	-	-	0/5V	DB-25	10
11	DIO 10	-	-	0/5V	DB-25	11
12	DIO 11	-	-	0/5V	DB-25	12
13	DIO 12	-	-	0/5V	DB-25	13
14	DIO 13	-	-	0/5V	DB-25	14
15	DIO 14	-	-	0/5V	DB-25	15
16	DIO 15	-	-	0/5V	DB-25	16
17	DIO 1	D.GND	GND (tanto de señal como alimentación)=A.GND 1	0/5V	DB-25	17
18	DIO 1	D.GND	GND (tanto de señal como alimentación)=A.GND1	0/5V	DB-25	18
19	DIO 1	+5V	Alimentación (hacia afuera) 1	0/5V	DB-25	19
20	DIO 1	+12V	Alimentación (hacia afuera) 1	0/5V	DB-25	20
21	DIO 1	BLANCO		0/5V	DB-25	21
22	DIO 1	AMARILLO		0/5V	DB-25	22
23	DIO 1	LILA		0/5V	DB-25	23
24	DIO 1	VERDE		0/5V	DB-25	24
25	DIO 1	AZUL		0/5V	DB-25	25

Figura 3.4 Relé de control de frenos.

Panel de conexiones

De especial interés práctico resulta localizar cada uno de los terminales emplazados en el panel de conexiones de la ECU (tal y como aparece en la figura 2.1b), para identificar los conectores y facilitar el servicio o actualización de la misma. Como puede apreciarse en la figura 3.5a, cada terminal se encuentra convenientemente etiquetado.



(a) Fotografía del panel de conexiones. **(b)** Esquema del panel de conexiones.

Figura 3.5 Vista trasera de la ECU.

Quedan de esta manera descritas las principales variables manejadas por la ECU. El siguiente paso será, a partir de toda la información presentada en el primer bloque, establecer los requisitos para una nueva configuración y buscar alternativas en cuanto a *hardware* y *software*.

4 Análisis de requisitos y de configuración de la nueva ECU

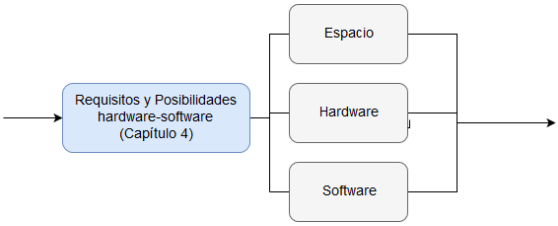


Figura 4.1 Esquema del segundo bloque.

Una vez estudiados todos los componentes y las variables que conforman la ECU, se decide que el sistema está próximo a alcanzar su tiempo máximo de vida operativa, y se considera la posibilidad de cambiarlos por otros más actuales y con mejores prestaciones que permitan continuar adelante con el proyecto. No obstante, antes de realizar cualquier propuesta de mejora de la ECU, es necesario plantear y recoger los requisitos exigibles a la nueva configuración.

4.1 Requisitos

En líneas generales, durante el desarrollo del presente trabajo se ha constatado que el proyecto del vehículo híbrido FOX comprende una alta dependencia de los continuos ensayos funcionales que se realizan en las instalaciones de la Escuela Técnica Superior de Ingeniería. Este hecho debería imponer a cualquier propuesta la capacidad de instalarse de la forma sencilla posible y de quedar completamente operativa en el menor tiempo posible, de modo que se puedan seguir realizando los ensayos descritos y validar de esta manera el trabajo en las distintas áreas (control de tracción, gestión de la energía...).

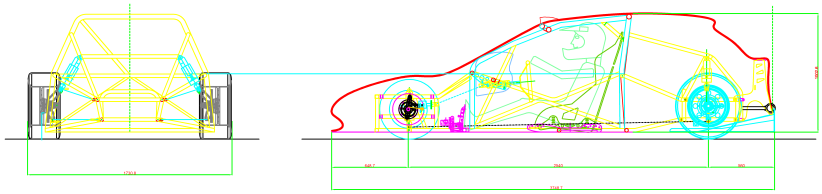
4.1.1 Requisitos topológicos

Una restricción importante es el espacio físico del que se dispone a bordo del vehículo FOX para albergar el sistema de control, ya que el chasis comercial que lo conforma tiene instalados los siguientes elementos:

En cualquier caso, el sistema de control debería ajustarse al espacio que ocupa actualmente el conjunto ECU-Supervisor, emplazado según se comentó en el capítulo 6.3.2

Cualquier otra alternativa de emplazamiento entraría en conflicto con la premisa de la sencillez y rapidez de instalación, puesto que para acomodar el sistema de control sería necesario inmovilizar el vehículo y utilizar herramientas especializadas para modificar el chasis. Estas modificaciones, además de suponer un coste adicional del proyecto (bien sea económico por la adquisición de tales herramientas o en tiempo por el estudio previo y el propio empleado en la modificación), comprenderían un cambio significativo en la dinámica del vehículo, en tanto que afectarían a la distribución de masas del mismo.

En cuanto a accesibilidad, el sistema debe poder ser fácilmente extraído de su emplazamiento para su prueba en bancada o para programar nuevas funcionalidades, además de para ser reparado en caso de fallo. En este aspecto, la ECU de partida se encuentra instalada con las conexiones orientadas hacia el espacio abierto entre los asientos del habitáculo, lo cual podría ser una buena solución para cualquier montaje propuesto.



Zona Trasera

- Pila de combustible
- Dos bombonas presurizadas de hidruros metálicos
- Conv. de potencia trasero
- Baterías

Habitáculo y zona delantera

- Pantalla BMS
- Pantalla supervisor
- Interfaz de usuario (HMI)
- Conv. de potencia delantero

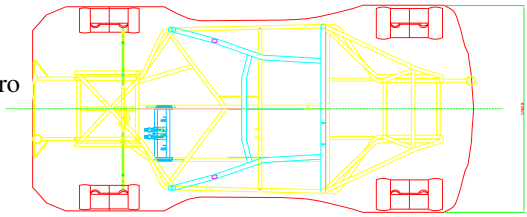


Figura 4.2 Proyecciones del chasis.

4.1.2 Requisitos funcionales

El objetivo principal del estudio de modernización de la ECU consiste en dotarla de componentes con prestaciones adaptadas a los entornos actuales de control de vehículos embarcados. No obstante, el sistema físico que se desea controlar (El vehículo FOX) seguirá disponiendo de las mismas variables de control (sección 3) procedentes de los sensores, al menos en el corto y medio plazos. Esto supone que el número de señales de entrada y salida tanto analógicas como digitales de la nueva configuración debe ser igual al actual o superior si se quieren reservar algunas para aplicaciones futuras. En este sentido resulta útil recapitular las necesidades de control del FOX.

Variables de salida analógicas 8 como mínimo, véase la tabla 3.2

Variables de entrada analógicas al menos 9, tabla 3.1

Variables de entrada/salida digitales 13 como mínimo, tabla 3.3

Además, con el objetivo de hacer rentable la inversión necesaria para costear la actualización de la ECU, cualquier característica funcional que se atribuya a la nueva configuración debe superar, o al menos igualar a las que posee en el nivel de partida. En base a esto se pueden analizar los requisitos que la nueva ECU debe satisfacer. Por ejemplo, un aspecto de importancia para la evaluación del nuevo sistema sería la capacidad de procesamiento de datos. Si se opta por un computador o SBC distinto al utilizado (*PCM 3370* descrito en 2.2.2), la elección debe hacerse de manera que el rendimiento en cuanto a velocidad y capacidad (en MHz y número de núcleos en el procesador) sean superiores a los dispuestos por el fabricante en referencia al mismo.

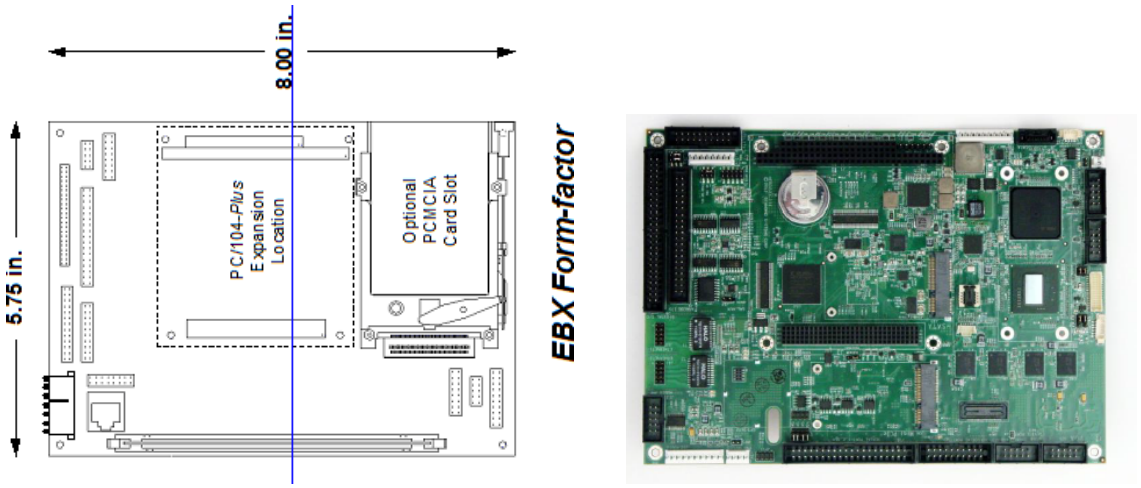
4.2 Hardware

En primer lugar se analiza la oferta comercial de *hardware* sobre los que se va a diseñar la nueva ECU. Conviene comentar que la variedad de modelos de ordenadores ideados para aplicaciones embarcadas es enorme en el mercado especializado, y que no obstante, muchos de ellos se han descartado por el hecho de no ser compatibles con las necesidades de control comentadas en la sección anterior; quedan fuera de la lista de posibles configuraciones aquellos que no disponen de capacidad para manejar el número de señales necesarias y que además no presentan posibilidad de expansión mediante módulos PC/104.

4.2.1 Hercules III de *Diamond Systems*

La primera opción a barajar es la de un ordenador de placa única (SBC, *Single Board Computer*) de *Diamond Systems* (fig. 4.3b).

El factor de forma EBX



(a) Factor de forma *EBX* . **(b)** Placa Hércules III .

Figura 4.3 Comparación entre el factor de forma *EBX* y la distribución de la placa *Hercules III*. Nótese la correspondencia espacial entre componentes y conectores en ambas imágenes..

EBX son las siglas de *Embedded Board, eXpandable*. Este estándar es el resultado de la colaboración entre las industrias líderes para unificar la computación embarcada en un único factor de forma compacto. El *EBX* combina una distribución estandarizada con interfaces abiertas. Es lo suficientemente pequeño para las aplicaciones embarcadas más exigentes, pero lo suficientemente grande como para contener todas las características de un ordenador: *CPU*, memoria, interfaces de almacenamiento masivo, controladores de *display*, puertos serie/paralelo, y otras muchas funciones.

Una de sus pricipales ventajas es que las posibilidades de expansión son muy flexibles, permitiendo añadir funcionalidades adicionales fácil y cómodamente. Este sistema de expansión está basado en algunos de los estándares industriales más populares: PC/104, PCI, PC/104-Plus, PCI-104, y PCMCIA. Véase la figura 4.3a. Algunas de las ventajas que ofrece este computador son:

Compacidad: Un SBC lleva integrados todos los principales componentes para su funcionamiento en aplicaciones embarcadas (fuente de alimentación, adquisición de datos, DAC, tarjeta CAN,...etc), lo cual es idóneo para reducir el número de componentes de la ECU. La placa *Hércules III*, además, posee la memoria RAM soldada, impidiendo así problemas de contacto entre las pistas de los conectores que se suelen dar en entornos con fuertes vibraciones.

Compatibilidad con el sistema operativo actual: Los sistemas operativos que soporta incluyen a los de tipo *UNIX*. Es por tanto completamente compatible con QNX 6.3.2..

No obstante, su aplicación conllevaría las siguientes problemas:

Necesidad de expansión: El principal elemento condicionante de un sistema de control electrónico a nivel práctico son las salidas analógicas. Como se indicó en el capítulo 3, la configuración inicial contempla el uso de 8 variables de tipo salida analógica, mientras que las disponibles en el sistema considerado son únicamente cuatro. Por tanto, una solución para suplir esta carencia sería expandir la placa conectando un módulo externo de E/S analógicas a través del puerto PCI disponible en el *Hercules III*, lo cual no supondría ninguna mejora a nivel de compacidad respecto del sistema actual. Si bien sería posible utilizar el módulo *Ruby-MM-1612*, la página oficial de *Diamond Systems* desaconseja su uso para nuevos diseños, proponiendo el empleo de la versión mejorada 1616.

Coste: El precio de mercado de este computador es bastante superior al combinado de todos los módulos PC/104 actuales, suponiendo un factor negativo frente a la implantación de este sistema como ECU.

4.2.2 SBC2596 de *Micro/Sys*

Se trata también de un SBC de factor de forma EBX, salvo que usa un bloque terminal extraible como conector de alimentación. Las características se presentan en el correspondiente *datasheet* en [11]:

Procesador: *Intel Pentium Tillamook* a 266MHz como máximo.

Entradas Analógicas: 32 canales de 16 bits.

Salidas analógicas: 4 canales de 16 bits.

Las ventajas de este SBC frente a la configuración actual son:

GPS: Incluye un GPS, lo cual podría suponer un ahorro de espacio al quedar ambos sistemas integrados en la misma unidad en lugar de ser independientes.

Relación calidad-precio: En caso de sustituir el hardware actual, este equipo presentaría la mejor relación calidad-precio de todos los considerados.

Y los inconvenientes:

Necesidad de expansión: Al igual que el *Hercules III*, dispone de cuatro salidas analógicas que harían necesaria su expansión mediante un módulo adicional.

Menor capacidad de procesamiento de datos: Como se ha indicado en las características principales, el CPU de esta placa posee unas prestaciones algo peores que el del *PCM-3770*.

Figura 4.4 EBX SBC 2596 ([11]).

4.2.3 *Copperhead* de *Versalogic corp.*

Consiste en un SBC con factor de forma *EBX* con capacidad para implementar distintas versiones de procesadores *Intel* de tercera generación. Esta característica lo convierte en la opción con mayor potencia en cuanto a procesamiento de datos. Las características favorables más relevantes en lo referente a su uso como ECU del FOX se presentan a continuación.

Procesador Según el *Datasheet* implementa la gama de procesadores *Intel* de tercera generación, con posibilidad de elegir entre las versiones *Core i7*, *Core i3* y *Celeron*.

Variables Dispone de 16 canales de entradas analógicas, 8 de salidas analógicas y 32 de E/S digitales, lo cual se adapta perfectamente a las necesidades de control dispuestas en la sección 4.1.2 sin necesidad de expansión.

Sistemas operativos soportados

A pesar de las anteriores características, las desventajas que presenta siguiendo los criterios expuestos al comienzo de esta sección son:

CANbus no incluido Aunque se especifica explícitamente la compatibilidad con módulos controladores de bus CAN, la placa no los incorpora, haciendo necesaria su expansión para incorporar esta funcionalidad.

Coste económico prohibitivo Al tratarse de una compacta y potente computadora destinada a soluciones de control embarcado muy exigentes su precio de venta rebasa con creces al de la configuración actual y es superior al de cualquiera de las alternativas presentadas en este capítulo.

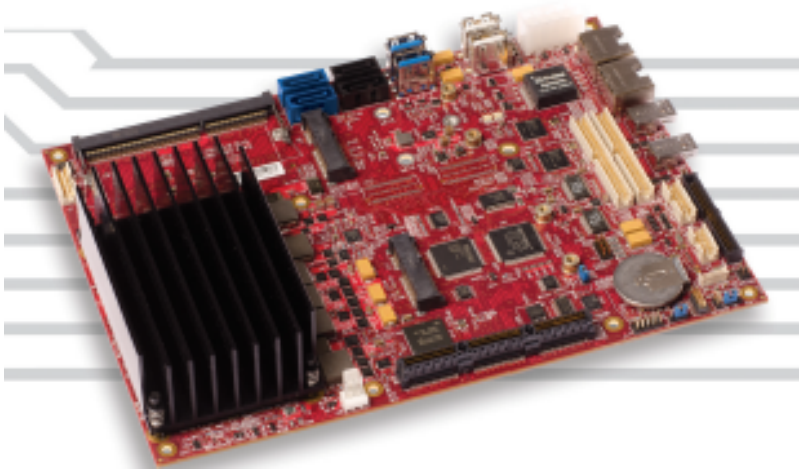


Figura 4.5 EBX *Copperhead*.

4.2.4 Módulo PC/104 con tarjetas de expansión

Cabe también la posibilidad de seguir empleando una tarjeta SBC de factor de forma PC/104 como pieza central del montaje de la ECU, y expandir sus funcionalidades con módulos adicionales de entradas/salidas analógicas y digitales y de comunicación CAN, de manera similar a la configuración de partida pero con componentes actualizados. Esta línea de pensamiento queda reforzada si se considera probar el sistema operativo *Linux* directamente en la configuración de *hardware* actual, conociendo la capacidad de un RTOS diferente en un sistema validado durante los últimos años y reduciendo la inversión inicial a prácticamente cero. Todo ello sirve de motivación para la realización de la prueba descrita en la sección 6.

4.3 Software

Seguidamente se proponen sistemas operativos para administrar el hardware de la ECU.

De entrada se descarta uno basado en *Windows*, puesto que las aplicaciones recomendadas por la compañía se encuentran más orientadas a interfaces en empresas que a entornos de control exigentes como el FOX. Además no existe la posibilidad de añadir funcionalidades RT a un SO *Windows* multipropósio, lo cual limita ampliamente las posibilidades que puede ofrecer en un entorno académico. Dicho esto, se estudian las opciones basadas en las soluciones más utilizadas en entornos embarcados.

4.3.1 Versión actualizada de QNX

Una de las principales bondades del sistema operativo inicial instalado en la ECU (La versión gratuita 6.3.2 de QNX analizada en el capítulo 2) es que resulta muy estable mientras se ejecuta, pese a que presenta problemas de corrupción de datos al introducir periféricos antes de las pruebas que se realizan. Además se trata de una versión desactualizada y sin soporte.

Figura 4.6 Logotipo de QNX.

La primera solución que se propone en lo referente a software es, por tanto, instalar una versión comercial de pago actualizada para remediar todos los fallos expuestos. La última varsión disponible en *http://www.qnx.com/download/* es la 7.0, aunque se podría estudiar la posibilidad de utilizar la 6.6 o la 6.5.x. Al tratarse de versiones de pago, la información referente a estos SO y a su desempeño se encuentra sujeta en

su mayoría al marketing de la empresa, y no abunda en la red.

4.3.2 Linux

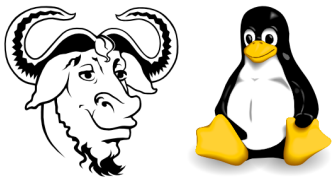


Figura 4.7 Logotipos de GNU/Linux.

La segunda alternativa corresponde a la instalación de otro sistema operativo basado en GNU/Linux. Paradigma del conocido como "Software libre" GNU (*GNU is Not UNIX*) es un sistema operativo de tipo UNIX desarrollado en 1983 por Richard Stallman, que utiliza un kernel Linux, ideado en 1991 por Linus Torvalds.

Algunas de las ventajas que presenta frente a otros SO son que *GNU/Linux* se encuentra disponible en una amplia gama de variaciones en la unión entre estas dos entidades, destinadas a satisfacer a grupos de usuarios determinados. Tales variaciones reciben el nombre de distribuciones. Éstas se encuentran disponibles gratuitamente en sus respectivas páginas junto a sus códigos fuente, de manera que son completamente presonalizables, implicando que el SO puede modelarse a conveniencia ante un cambio en las necesidades de control.

Todas estas particularidades hacen de *Linux* la mejor alternativa alrededor de la cual diseñar el nuevo software de la ECU. No obstante el sistema debe presentar una serie de características singulares a la luz de lo explicado a continuación.

Durante la operación del vehículo, la ECU se encuentra ejecutando continuamente procesos de adquisición de datos procedentes de los sensores y enviando las señales dictadas por los programas de control a los motores. Imagínese que durante este intervalo el sistema operativo comenzara a ejecutar un proceso de baja prioridad no relativo a la tarea de control del vehículo. Puede ser entonces que su comportamiento fuera poco preciso o incluso impredecible, debiendo evitarse esta situación a toda costa. Esto sólo se puede conseguir de manera estricta empleando un SO que ofrezca la posibilidad de ejecutar las tareas de mayor importancia de forma determinista, conocido como *Sistema Operativo en Tiempo Real o RTOS*, estudiado en la sección siguiente.

5 Sistemas operativos en tiempo real

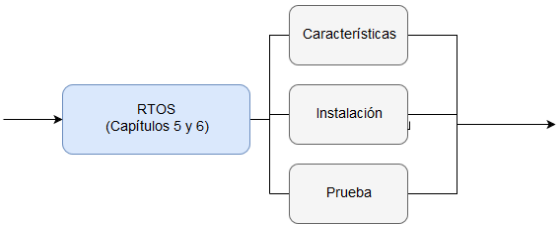


Figura 5.1 Esquema del tercer bloque.

5.1 Introducción

Los sistemas operativos en tiempo real (*Real Time Operating Systems, RTOS*) proporcionan soporte básico para la gestión de recursos, sincronización, comunicación y manejo de entradas y salidas. Este tipo de sistemas operativos han evolucionado desde los especializados en una única tarea hasta un amplio rango más generalista (como las variantes en tiempo real de *Linux*).

En cuanto al nivel de tiempo real, y con el objetivo de construir un contexto para esta sección, una clasificación general puede ser la siguiente:

Duro o *hard* No alcanzar un tiempo de ejecución prefijado es considerado como un fallo total del sistema. Es el más estricto de los tres.

Firme (*firm*) No se considera un fallo el no alcanzar las restricciones temporales de manera esporádica, aunque esto pueda empeorar la calidad del servicio.

Suave o *soft* La utilidad de un resultado se degrada tras su tiempo de ejecución preestablecido, deteriorando con ello la calidad del servicio.

En este aspecto, también se ha visto una evolución desde los RTOS completamente predecibles y capaces de soportar aplicaciones críticas de seguridad hasta los que proporcionan servicios de tiempo real *soft*. Este soporte incluye el concepto de calidad del servicio (QoS), a menudo aplicado tanto a multimedia como a RTOS más complejos.

Los investigadores en el campo de los RTOS han desarrollado nuevas ideas y paradigmas que mejoran los sistemas operativos tradicionales para que sean más eficientes y predecibles. Algunas de estas ideas se encuentran en los sistemas convencionales y muchas otras en una amplia variedad de RTOS del mercado actual, que incluye *kernels propietarios*, y versiones en tiempo real de sistemas operativos populares como *Linux* (véase el siguiente capítulo) y *Windows NT*. Muchos estándares de la industria se han visto influenciados por la investigación de RTOS, incluyendo las extensiones en tiempo real de *POSIX*, la especificación en tiempo real de *Java*, *OSEK* (estándar de RTOS de automoción), *Ada83* y *Ada85*.

5.1.1 Sistemas operativos

Antes de estudiar este tipo de SO con mayor profundidad conviene recordar la definición de sistema operativo y el papel que el *kernel* desempeña en este contexto.

Se define sistema operativo como el conjunto de *software* que gestiona la conexión entre las capas de usuario de más alto nivel con las de más bajo nivel o de *hardware*. Entre la amplia variedad de campos que maneja se encuentran la memoria, las entradas y salidas, el sistema de archivos...etc. (figura 5.2). La capa de más bajo nivel de un SO que gestiona el *hardware* de forma directa recibe el nombre de núcleo o *kernel*, y por ello presenta una gran importancia a la hora de caracterizar el rendimiento de un RTOS.

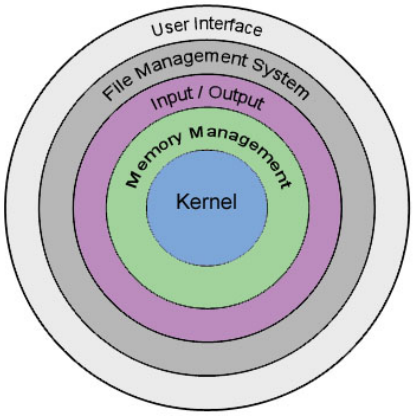


Figura 5.2 Ilustración de las distintas capas de un SO.

5.1.2 Nociones sobre latencia

La latencia de un sistema se define como el retraso entre el tiempo de entrada a un sistema y el de salida o , en términos más sencillos, el intervalo de tiempo transcurrido entre el estímulo del sistema y su respuesta. Si este intervalo además presenta variaciones en el tiempo, éstas reciben el nombre de *jitter*.

En aplicaciones en tiempo real, interesa que ambas magnitudes temporales sean lo más pequeñas posible. De esta manera, latencias bajas asegurarán que se cumplen los márgenes temporales establecidos, y un *jitter* bajo garantizará el grado de determinismo del sistema. En la figura 5.3 se ilustran ambos conceptos.

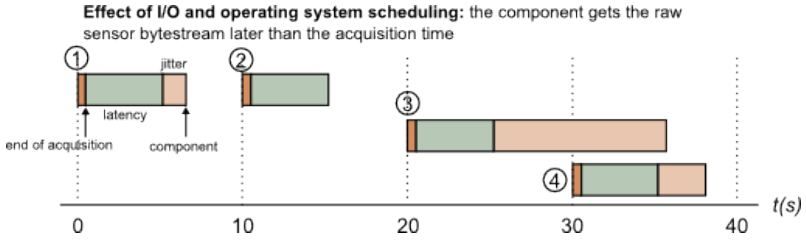


Figura 5.3 Latencia y *jitter* ([12]).

5.2 Clasificación y arquitectura de los RTOS

Los sistemas operativos en tiempo real hacen énfasis en la predictibilidad y eficiencia, y poseen la capacidad para soportar restricciones en cuanto a tiempos de ejecución. Existen varias categorías generales de RTOS: pequeños como los *kernels propietarios*, extensiones en tiempo real para sistemas operativos de tiempo compartido convencionales como *UNIX/Linux*, en *QoS* y de investigación universitaria. En la presente sección se tratarán los dos primeros por ser los más empleados en el ámbito de la computación embarcada.

5.2.1 *Kernels* propietarios o privativos

Están concebidos para soportar directamente aplicaciones de tiempo real. Son ligeros y se utilizan a menudo para pequeñas aplicaciones embarcadas cuando se requiere garantizar una ejecución rápida y predecible, estando normalmente altamente especializados para la aplicación concreta a la que están destinados. El simple coste de desarrollar y mantener un núcleo, así como la calidad que ofrecen las alternativas convencionales modificadas, han supuesto una reducción en su uso y frecuentemente la suspensión del soporte. Además presentan el inconveniente de que utilizan *drivers* de dispositivos específicos que dificultan su modificación o actualización. Para conseguir la velocidad y la predictibilidad necesarias, los *kernels* son versiones recortadas y optimizadas de sistemas operativos de tiempo compartido. Con todo ello, para conseguir los objetivos expuestos, el *kernel* debe:

- tener un interruptor de contexto (context switch) rápido,
- ser pequeño (con funcionalidad mínima),
- responder a interrupciones externas rápidamente (algunas veces con una latencia máxima para enviar un evento garantizada, pero, generalmente no se da garantía del tiempo en el que se completará el evento; esta última se puede computar si las prioridades se asignan correctamente),
- minimizar los intervalos durante los que las interrupciones se desactivan
- proporcionar particiones fijas o variables para gestionar la memoria, así como tener la habilidad de bloquear código y datos en la memoria,
- proporcionar archivos secuenciales que puedan acumular información a alta velocidad.

Para cumplir estos requisitos temporales, el *kernel*:

- soporta *multitarea*,
- proporciona un mecanismo de temporización con derecho preferente,
- asegura un tiempo acotado de ejecución para la mayoría de tareas primarias,
- mantiene un reloj de alta resolución en tiempo real,
- dispone de alarmas especiales y mensajes de fin de ejecución,
- soporta disciplinas de cola tales como resolver primero las tareas con tiempo límite más corto y primitivas para enviar un mensaje al principio de una cola.
- Dispone de tareas primitivas para retrasar el procesamiento una cantidad fija de tiempo y suspender/retomar la ejecución.

En general los núcleos también desarrollan comunicación *multi* e *intertarea* y sincronización a través de tareas primarias como *buzones* (colas de mensajes), *eventos, señales, mutex y semáforos*. Mientras que éstas últimas están diseñadas para ser rápidas, tal término es relativo e insuficiente para referirse a restricciones en tiempo real. Sin embargo, muchos diseñadores de RTOS las utilizan como base para construir tales sistemas. Esto ha resultado efectivo en pequeñas aplicaciones embarcadas, que al ser simples permiten saber con relativa sencillez si se cumplen las restricciones temporales. Consecuentemente, los *kernels* proporcionan exclusivamente la mínima funcionalidad que se necesita.

5.2.2 Extensiones en tiempo real para OS convencionales

Un segundo enfoque a los RTOS consiste en extender productos convencionales o multipropósito, como por ejemplo *Unix* a *RT-Unix*, *Linux* a *RT-Linux*, etc. Las versiones en tiempo real de los sistemas operativos son generalmente más lentas y menos predecibles que los *kernels* propietarios, pero poseen más funcionalidades y mejores entornos de desarrollo de *software*, consideraciones muy importantes en muchas aplicaciones grandes o complejas.

Otra ventaja significativa es que se basan en un conjunto de interfaces o estándares familiares que facilitan la portabilidad. Para *Unix*, puesto que existen muchas variaciones, el organismo *IEEE* desarrolló un esfuerzo por estandarizarlas, sintetizándose en un conjunto común de interfaces a nivel usuario llamado *POSIX*. El esfuerzo se centró en once funciones relacionadas con el tiempo real: *temporizadores, programación de prioridades, memoria compartida, archivos en tiempo real, semáforos, comunicación entre procesos, notificación de eventos asíncrona, bloqueo de memoria de procesos, entradas y salidas síncronas y asíncronas e hilos*.

Existen varios problemas cuando se intentan convertir los sistemas operativos convencionales a sus versiones en tiempo real. Dichos problemas pueden existir tanto en la interfaz del sistema como en la implementación, incluyendo un margen intolerable y excesiva latencia al responder a las interrupciones, en parte debido a la falta de derecho preferente del kernel y a las colas FIFO internas. Estos y otros problemas de hecho han sido

resueltos para resultar en un sistema operativo que procesa tanto de forma convencional como en tiempo real, como es el caso del utilizado en este trabajo.

Las capacidades de tiempo real se pueden añadir a los sistemas operativos convencionales de distintas maneras, agrupadas en las siguientes categorías:

Compliant kernel: En este enfoque, se modifica un sistema en tiempo real existente para que los archivos binarios de *Linux* puedan ejecutar sin modificarlos. En esencia, la funcionalidad y la semántica de las llamadas deben ser emuladas por el sistema operativo nativo.

Kernel dual: Se sitúa un *kernel* en tiempo real "duro" pero de pequeño tamaño debajo del sistema operativo nativo (como *Linux*, ver figura 5.4), para que capture todos los accesos y las interrupciones desde el *hardware* subyacente. Este *kernel* pequeño programa varias tareas en tiempo real situadas en su entorno y ejecuta el sistema operativo nativo como su tarea de mínima prioridad. Como resultado, las aplicaciones nativas se pueden ejecutar sin cambios, mientras que las tareas en tiempo real "duro" pueden conseguir altos rendimiento y predictibilidad. También se proporciona un nivel de comunicación (sin ser RT) entre el núcleo en tiempo real y el nativo para intercambiar información.

Los contras de este enfoque es que no existe protección de memoria entre las tareas en tiempo real y el núcleo nativo, por lo que el fallo de cualquier tarea en tiempo real puede ocasionar una caída total del sistema. El *kernel* en tiempo real fino también necesita su propio conjunto de *drivers* para funcionar en tiempo real.

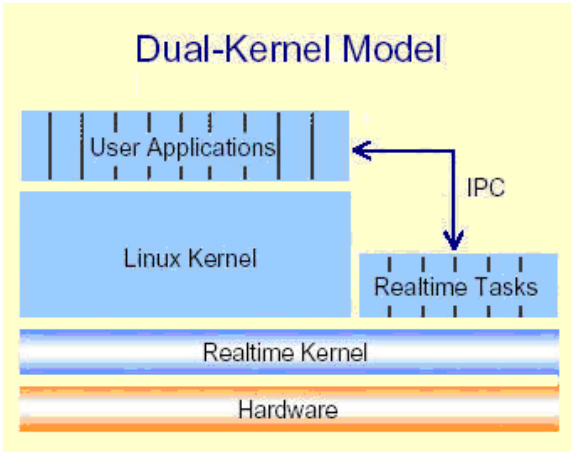


Figura 5.4 Modelo de kernel dual ([13]).

Modificaciones del *kernel*: se realizan cambios en el núcleo del *kernel* para hacerlo lo suficientemente predecible y determinista para que se comporte como un RTOS. Para ello se puede:

- Usar programación de prioridad fija con un programador O(1).
- Emplear temporizadores de alta resolución, dotar al *kernel* de derecho de prioridad (de tal manera que a un proceso de más baja prioridad debido a una llamada en curso del sistema se le pueda dar derecho de prioridad a través de otro de mayor prioridad que pueda ser elegido para ejecutarse).
- Habilitar el soporte de protocolos de herencia de prioridad para minimizar la inversión de la misma.
- Hacer los interruptores programables utilizando hilos del *kernel*.
- Utilizar procesos periódicos.
- Reemplazar las colas FIFO (*First In First Out*) por colas de prioridad.
- Optimizar las rutas más largas a través del *kernel*

Enfoque del Resource *kernel*: Se extiende el núcleo con el fin de proporcionar soporte para reservas además de para la aproximación tradicional de programación de prioridad fija con derechos. Esta última puede ocasionar problemas cuando un proceso de prioridad relativamente alta excede su tiempo de ejecución esperado o entra en un bucle infinito. Los *Resource kernels* soportan y refuerzan la reserva

de recursos, tales como que ninguna tarea que se comporte de manera errónea pueda impactar en el comportamiento temporal de otra.

5.3 Paradigmas

Los RTOS utilizan varios paradigmas. Los conceptos clave incluyen las garantías de tiempo real estricto y permisivo, el control de admisión y la reflexión. Muchos de estos conceptos trabajan juntos para conformar el paradigma que ofrece un *kernel* en particular.

Garantías En general, los *kernels* más pequeños y deterministas proporcionan soporte para sistemas estrictos en cuanto a tiempos de ejecución. En ellos, todas las entradas y detalles del sistema son conocidos, y un cuidadoso análisis y diseño pueden resultar en el cumplimiento de los requisitos en tal aspecto. Realizando el análisis es también posible tener en cuenta los márgenes del *kernel*. Los sistemas de tiempo real estricto críticos para la seguridad emplean típicamente márgenes confortables para la utilización de recursos (tales como asegurar que el uso total de un recurso no excede el 50-60 %).

Los más grandes y dinámicos soportan sistemas de tiempo real permisivo. Aquí las garantías del QoS están definidas y se deben cumplir en sentido probabilístico.

Control de admisión El control de admisión es una función que decide si se permite o no el acceso de nuevo trabajo al sistema. Incluye un modelo del estado de los recursos, información de la petición entrante, el algoritmo exacto para realizar el control y los protocolos a seguir tras la admisión o el rechazo.

Muchos de los RTOS estrictos están programado estáticamente y operan de forma altamente determinista. Esto facilita el análisis de temporización requerido por éstos y no existe la noción del control de admisión. Sin embargo, muchos RTOS operan en ambientes dinámicos donde la programación estática es muy costosa o rígida. Lo que se requiere es una solución que permita un cuidadoso análisis *on-line* de temporización y programación dinámica.

Reserva de recursos Se trata del acto de asignar parte de los recursos del sistema a una tarea. En el afán por mantener un nivel razonable de multi-programación. Los primeros sistemas operativos no incluían esta característica.

Reflexión La información reflexiva está compuesta por *metadatos* del sistema. Pueden hacer referencia a una aplicación o sobre las propiedades y el rendimiento del SO. Todos los sistemas operativos son reflexivos en mayor o menor grado. No obstante si se eleva el concepto de reflexión a un principio central del sistema, entonces es posible construir sistemas más flexibles, y centrarse en la necesidad de elegir la información necesaria correcta para un sistema dado. Parte de esta flexibilidad se puede utilizar para obtener un mejor rendimiento para alcanzar las restricciones temporales, por ejemplo.

Una vez introducidas las características de un RTOS no es difícil comprender la necesidad que un proyecto tan exigente en cuanto a control como el FOX tiene de implantar uno. En concreto, y con vistas a conseguir un SO con la mayor versatilidad y soporte *online* posible, se puede afirmar que una de las opciones más interesantes en cuanto al *software* es la extensión de tiempo real para un SO basado en *GNU/Linux*, como se ha explicado en la sección 5.2.2. En el siguiente capítulo se estudiará una implementación práctica de dicha solución.

6 Implementación de Linux RT

En las siguientes secciones se describirá el proceso seguido para instalar y monitorizar un sistema operativo en tiempo real basado en *Linux* en un computador con factor de forma PC/104.

6.1 Material disponible

Las pruebas se realizaron en un ordenador PC/104 modelo *VDX104* de *Tri-M Technologies, inc.* facilitado por el Departamento de Ingeniería Aeroespacial y Mecánica de Fluidos de la Escuela Técnica Superior de Ingenieros de la Universidad de Sevilla. Entre los materiales cedidos se encuentran los siguientes (figura 6.2a):

- Computador *VDX104*, con características (véanse [14] y la figura 6.2b para información adicional):

  Factor de forma *PC104+*

  CPU *Vortex86-DX-800MHz*

  Puerto serie 4 terminales *RS-232*

  Puerto LAN 10/100 Ethernet

- Módulo de fuente de alimentación *PC/104 HE 104-DX*, con rango de tensión de entrada comprendido entre 6VDC-40VDC.
- Transformador AC/DC *ATX-200* de 475W con ventilador de refrigeración, conectores para *fan* externo y posibilidad de proporcionar ±3.3VDC, ±5VDC, ±12VDC.
- Tarjeta *Compact Flash* de tipo I con capacidad de 4GB, con la distribución *slackware 13.0.0.0* de *GNU/Linux* instalada por el personal del departamento.
- Cable de red *Ethernet Cat5e*

Asimismo se fabricó un cable de comunicación puerto serie en el laboratorio del FCCL a partir de dos conectores macho *RS-232 DB9* y conductor de tipo cinta.

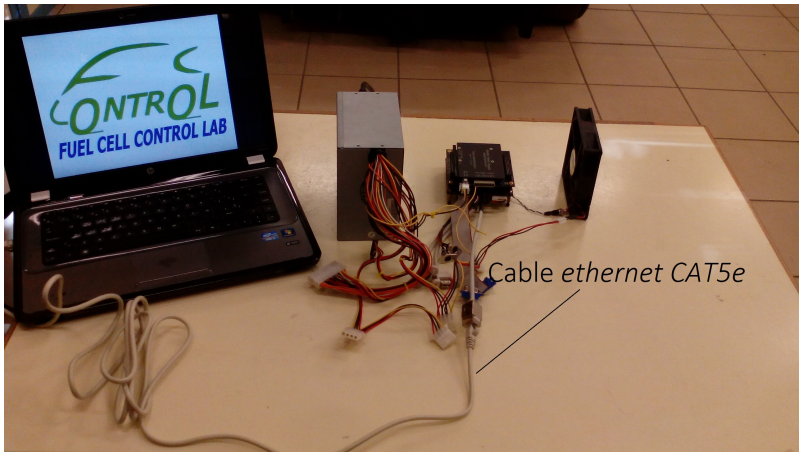
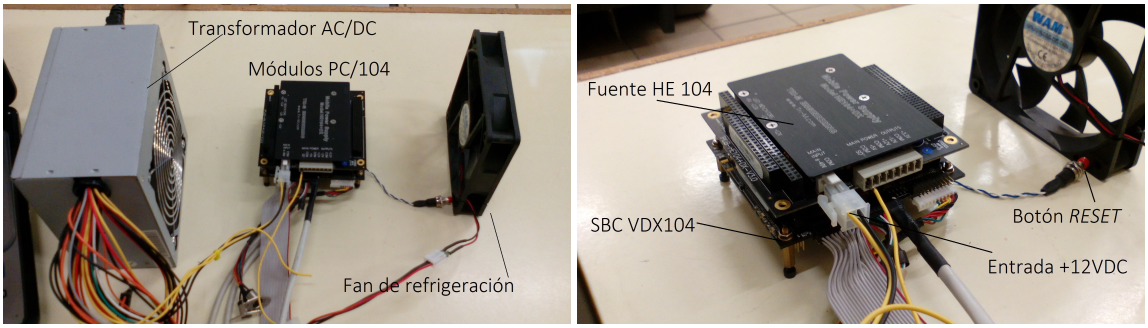


Figura 6.1 Vista general de la mesa de laboratorio.



(a) Material cedido por el departamento de Ingeniería Aeroespacial. **(b)** Detalle de los módulos PC/104.

Figura 6.2 Elementos de la prueba.

6.2 Montaje de laboratorio

6.2.1 Alimentación

El transformador AC/DC se encuentra conectado al módulo de alimentación en dos puntos; el primero se trata de la toma principal, capaz de aceptar tensiones en el rango de 6-40 VDC y compuesta por una ficha de conexión con dos terminales: uno de color negro (masa) y otro de color amarillo (activo), entre los cuales se ha medido una tensión de +12VDC mediante un voltímetro digital. Todo ello se puede visualizar en la figura 6.2b. El segundo, etiquetado con SD, consiste en una entrada de encendido y apagado opto-acoplada que controla las salidas de la fuente, conectada a la toma principal según se recomienda en el manual del fabricante en [15].

6.2.2 Comunicación remota

La comunicación con el computador PC/104 se debe hacer de forma remota, con otro equipo al que sí se le pueda instalar un monitor. Se hace necesario por tanto disponer de un terminal o *shell* remoto en dicho equipo. Los métodos para disponer de ella se explican a continuación.

Puerto serie Un puerto serie está basado en un conector de tipo *RS-232 DB9*. Este método es el más rápido y sencillo de llevar a cabo, con el único inconveniente de ser difícil encontrar PCs modernos que incorporen tal terminal. Además, tal y como se explicará más adelante, se trata de la única forma en que se puede acceder al *shell* si no se ha configurado la conexión *SSH* del dispositivo. Una posible alternativa consistiría en utilizar adaptadores *RS232-USB*, aunque no se ha considerado en este proyecto por falta de disponibilidad. La conexión entre los computadores se realiza tras configurar el adaptador del puerto con los siguientes valores:

- Puerto: COM1
- Velocidad (*Baud rate*): 115200 baudios o símbolos transmitidos por segundo
- Bits de datos: 8
- Bits de parada: 1
- Paridad y control de flujo: desactivados

Comunicación *SSH* *SSH* son las siglas de *Secure SHell*, un protocolo de comunicación que garantiza seguridad en las transmisiones entre equipos a través de un medio no seguro. Básicamente la información intercambiada es encriptada y desencriptada sin que el usuario tenga noticia de tal proceso. El medio de transmisión de datos puede ser cableado (cable *ethernet cat5e*) o inalámbrico. Desde el punto de vista práctico el más adecuado para acceder mediante la mayor parte de los ordenadores portátiles modernos, favoreciendo la flexibilidad en cuanto a dispositivos de acceso. Es por esto que, aun presentando mayor dificultad en la configuración que en el caso puerto serie, se dará prioridad a este tipo de conexión, aunque ambas son igualmente válidas.

6.3 Guía de conexión

6.3.1 Puerto serie

Considérese la primera vez que se conecta la placa a un PC sin que ésta posea una dirección *IP* asociada para la conexión *SSH* mediante *ethernet*. En esta situación, si se desea acceder a través de *SSH* es necesario establecer la conexión puerto serie y configurar la red *eth0* (Ver la sección 6.3.2). A continuación se describe de forma breve el proceso, según el sistema operativo del que se disponga. Para esta prueba se han utilizado los sistemas Windows 7 y Ubuntu 16.10.

Windows

La herramienta principal para acceder al terminal desde *Windows* consiste en el programa PuTTY, un cliente de *SSH* y *telnet*, desarrollado originalmente por Simon Tatham. PuTTY es un software de código abierto (se encuentra disponible en la red para cualquier usuario de forma gratuita) mantenido por un grupo de programadores voluntarios.

Como funcionalidad adicional, el programa también permite la conexión puerto serie, necesaria para configurar en primera instancia la red *ethernet* para su posterior conexión a través de *SSH*. Además, permite transferir archivos entre servidor y cliente mediante su propio protocolo de transmisión segura de archivos *Putty Secure File Transfer Protocol*.

Primeramente se configura el adaptador puerto serie del equipo con los datos que se han indicado y se descarga PuTTY desde *www.putty.org*. Tras la instalación se mostrará la ventana principal mostrada en la figura 6.3.

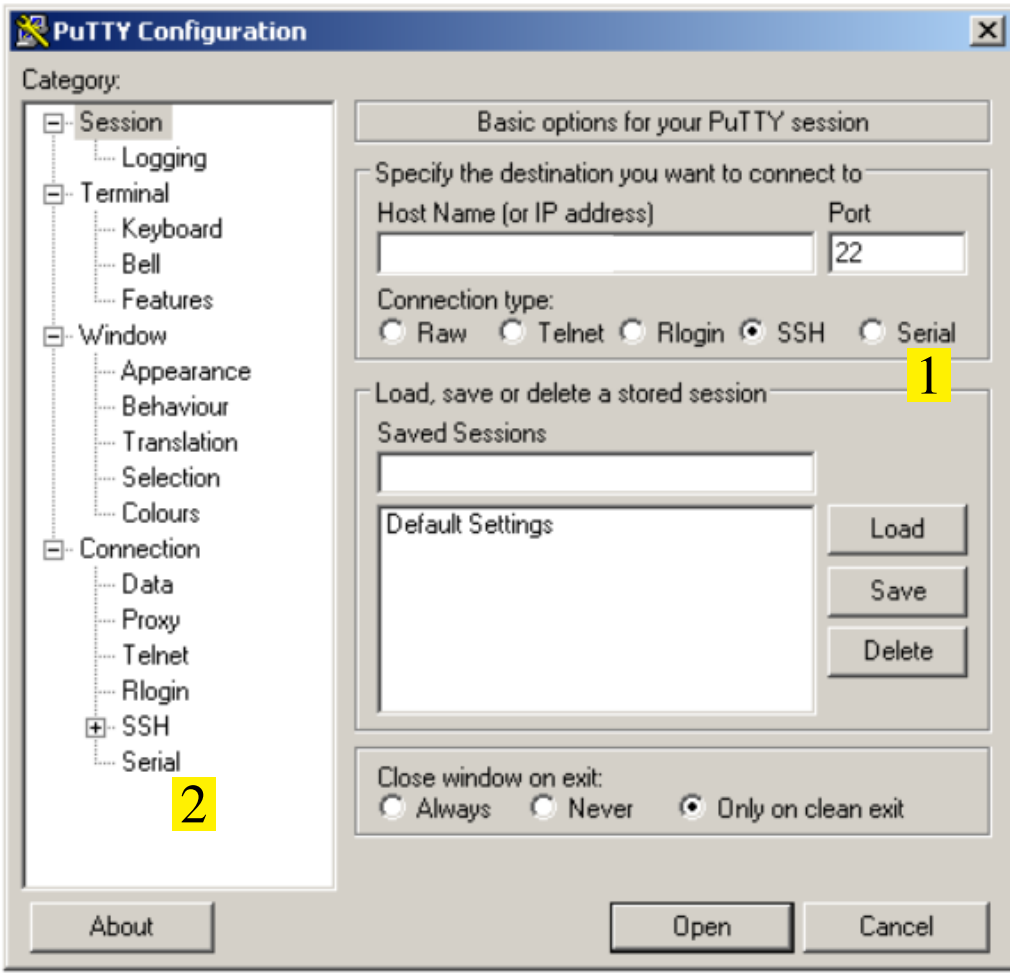


Figura 6.3 Ventana principal de PuTTY.

En la parte superior, etiqueta número 1, se encuentran las opciones correspondientes al tipo de conexión. Previamente a abrir la sesión, será necesario seleccionar el cuadro de diálogo *Serial*.

En la zona inferior izquierda de la ventana principal de PuTTY (etiqueta número 2) se encuentra la pestaña asociada a la configuración del puerto serie, cuyos campos deberán modificarse según lo dispuesto en la sección 6.2.2.

Una vez configurada la conexión, se puede iniciar mediante la pestaña *Open* situada en la parte inferior derecha de la ventana principal. Es en este punto cuando se puede encender el PC/104 conectado a través de puerto serie. El *shell* aparecerá entonces mostrado por pantalla. En caso contrario, se alcanzará tiempo máximo de espera para establecer la conexión de PuTTY, informándose de ello al usuario.

Linux

Para conectar la placa a un PC con *Linux* a través de puerto serie, es necesario configurarlo sin más que utilizar el siguiente comando en la terminal (como usuario *root*):

Código 6.1 Conexión puerto serie.

```
screen /dev/ttyS0 115200 -cs8 -cstopb -parenb #El directorio ttys0 hace
    referencia al primer puerto serie del equipo; téngase en cuenta si se tiene
     más de un dispositivo conectado a través de puerto serie.
```

6.3.2 Configuración para conexión *SSH* en el PC/104

Para poder conectarse a la placa mediante *SSH* es necesario que la red *eth0* de la placa se encuentre configurada adecuadamente, para lo cual se deberá acceder al mismo a través de puerto serie. Existen varios ficheros en el sistema de archivos de *Linux* relacionados con la configuración de las redes, entre los que se encuentra */etc/rc.d/rc.inet1.conf*:

Código 6.2 Modificando el archivo /etc/rc.d/rc.inet1.conf.

```
root@darkstar:~# vim /etc/rc.d/rc.inet1.conf #Se modifica el archivo mediante
    vim
...
# Config information for eth0:
IPADDR[0]="192.168.1.10" #Se asigna una dirección IP a la red ethernet eth0
NETMASK[0]="255.255.255.0" #y una máscara de red
USE_DHCP[0]="no"
DHCP_HOSTNAME[0]=""
...
# Default gateway IP address:
GATEWAY="192.168.1.1" #Puerta por defecto
```

Como puede observarse, la principal configuración necesaria es asignar una dirección IP al PC/104 que se utilizará para acceder mediante SSH a través de ethernet. Para verificar la configuración se ejecuta el comando *ifconfig*

Código 6.3 Comando ifconfig.

```
root@darkstar:~# ifconfig
eth0      Link encap:Ethernet HWaddr 00:07:1e:86:01:17
inet addr:192.168.1.10 Bcast:192.168.1.255 Mask:255.255.255.0
inet6 addr: fe80::207:1eff:fe86:117/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:1572 errors:0 dropped:0 overruns:0 frame:0
TX packets:1010 errors:0 dropped:0 overruns:0 carrier:0
```

```
collisions:0 txqueuelen:1000
RX bytes:144665 (141.2 KiB) TX bytes:161729 (157.9 KiB)
Interrupt:5

lo       Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/128 Scope:Host
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:4 errors:0 dropped:0 overruns:0 frame:0
TX packets:4 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:392 (392.0 B) TX bytes:392 (392.0 B)
```

El primer bloque que devuelve muestra la configuración de la red eth0, correspondiente a ethernet, y el segundo al bucle interno de software de la máquina. Puede observarse que la dirección IP que aparece junto al campo *inet addr* corresponde a la que se encuentra en /etc/rc.d/rc.inet1.conf. En caso de que sólo apareciese la red *lo* (situación posible únicamente en el caso de conexión a través de **puerto serie**) se deberá escribir en el terminal

Código 6.4 habilitando la red eth0.

```
root@darkstar:~# ifconfig eth0 up
#Si se quiere actualizar la dirección IP tras modificar /etc/rc.d/rc.inet1.conf
     PRECAUCIÓN: no utilizar las siguientes líneas mientras se esté conectado a
     través de SSH
root@darkstar:~# ifconfig eth0 down
root@darkstar:~# ifconfig eth0 up
```

También es conveniente reinicializar el servidor OpenSSH del PC/104 mediante la orden

Código 6.5 Órdenes principales del servidor SSH.

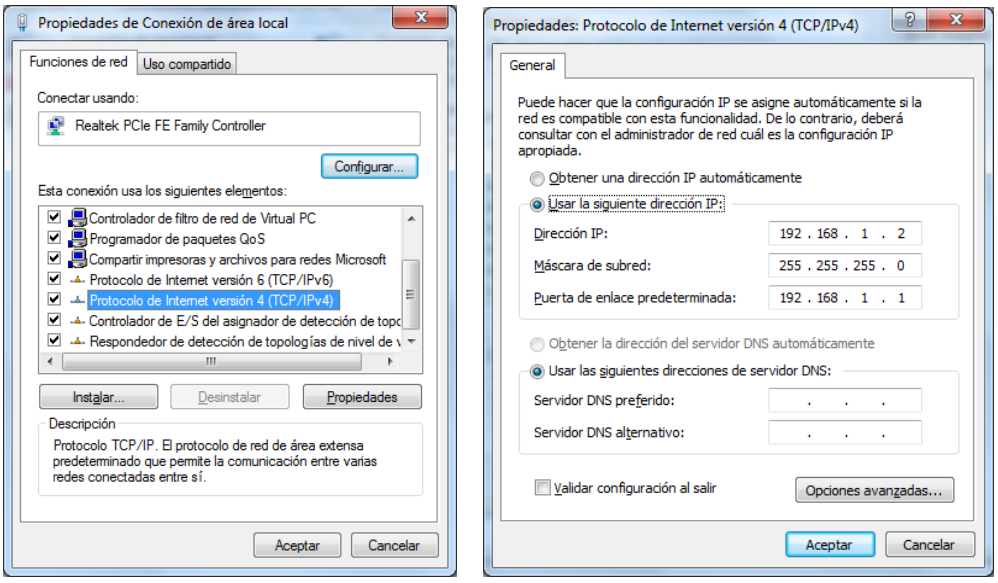
```
root@darkstar:~# /etc/rc.d/sshd stop # Parar el servidor SSH
root@darkstar:~# /etc/rc.d/sshd start # Inicializar el servidor SSH
root@darkstar:~# /etc/rc.d/sshd restart # Reinicializar el servidor SSH
root@darkstar:~# /etc/rc.d/sshd status # Comprobar el estado del servidor SSH
```

6.3.3 SSH

En este punto se está en condiciones de configurar y establecer una conexión *SSH* desde cualquier ordenador del laboratorio, tal y como se detalla a continuación.

Windows/PuTTY

Es necesario configurar el adaptador de red del equipo del laboratorio, que en el caso de esta prueba es el correspondiente a la red cableada al no disponerse de tarjeta adaptadora de red inalámbrica en el PC/104. Véase la figura 6.4.



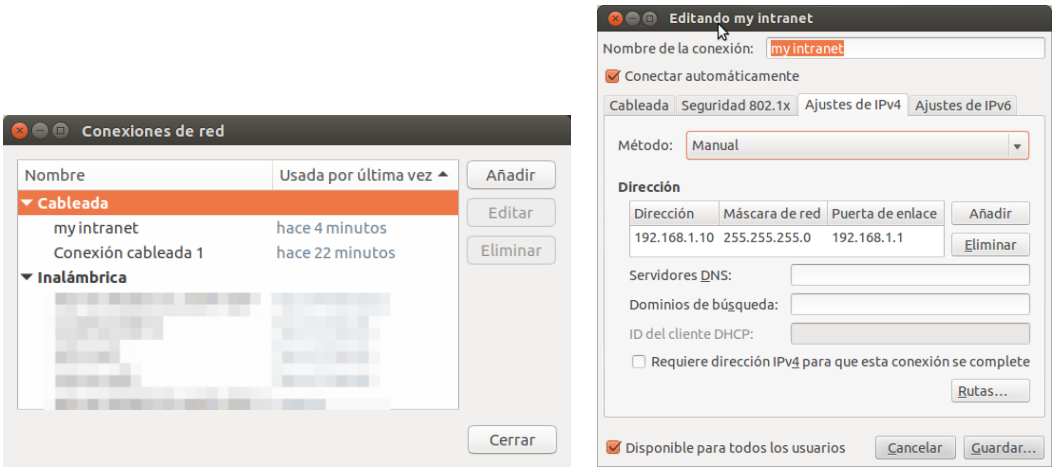
(a) Opciones del adaptador de red. **(b)** *TCP/IPv4*.

Figura 6.4 Configuración del adaptador de red de *Windows*.

Para conectarse al PC/104 se abre PuTTY y se escribe la dirección IP estipulada en la sección anterior y se abre la sesión *SSH* con la pestaña *Open*. El *shell* debería aparecer por pantalla mostrando el menú de autenticación.

Linux

El comando básico de Linux para establecer una conexión *Secure SHell* es *ssh*. Previamente a introducirlo en el terminal, y de forma similar a Windows, se crea una nueva red cableada y se modifica con los datos proporcionados por el *ssh daemon OpenSSH* instalado en la placa.



(a) Listado de redes en Linux. **(b)** Configuración de red cableada.

6.4 Transferencia de archivos

La primera tarea tras acceder al sistema es conocer la versión del SO y del kernel. Todo ello se puede conseguir sin más que ejecutar los siguientes comandos en la terminal:

Código 6.6 Datos relevantes.

```
root@darkstar:~# cat /etc/slackware-version #Versión del sistema operativo
Slackware 13.0.0.0.0
root@darkstar:~# uname -r #Versión del kernel
2.6.29.6
```

Con este último dato es posible descargar la versión correcta de las fuentes y del parche RT. En primer lugar se descarga el código fuente de la versión 2.6.29.6 del kernel de Linux de *kernel.org* ([16]) con el ordenador desde el que se está accediendo al PC/104. De entre todos los enlaces disponibles para una versión determinada se debe seleccionar aquella en formato .tar.bz2.

6.4.1 Linux

Para transferir el archivo se utiliza el comando sftp *(*Secure File Transfer Protocol) en el ordenador del banco, asegurándose previamente que se es usuario root del mismo para obtener todos los permisos necesarios. Es posible realizar el proceso en dos terminales abiertas simultáneamente (para abrir una sólo es necesario presionar la combinación de teclas *ctrl*+*alt*+*t*); en una de ellas se realiza la transferencia y en la otra se accede al shell del PC/104.

Código 6.7 Terminal 1: Conexión via sftp.

```
root@server:~# sftp root@192.168.1.10
root@192.168.1.10's password:
Connected to 192.168.1.10.
sftp>
```

A continuación se debe determinar el directorio donde se desea transferir el archivo para lo cual se utiliza el comando *cd*. Para disminuir el número de pasos a realizar, se transferirá a /usr/src.

Código 6.8 Transferencia del kernel 2.6.29.6.

```
sftp> cd /usr/src
sftp> put /ruta/del/archivo/del/kernel.tar.bz2
sftp> bye #Cierra la transferencia sftp
```

Una vez el archivo se encuentra en la placa se procede a extraerlo en el mismo directorio

Código 6.9 Terminal 2: Extraer el kernel.

```
root@darkstar:~# cd /usr/src
root@darkstar:/usr/src# tar -C /usr/src -jxvf linux-2.6.29.6.tar.bz2
#comprobamos que se ha creado una carpeta con el kernel en el directorio
    deseado:
root@darkstar:/usr/src# ls -a
./  ../  linux-2.6.29.6/ xenomai-2.5.3/
```

Con el kernel ya extraído, el siguiente paso es transferir el parche de tiempo real *rt-24* ([17]) mediante la terminal 1 dentro de la carpeta del código fuente recién extraída. La sintaxis es la misma que en el caso anterior

Código 6.10 Terminal 1: Transferencia del parche 2.6.29.6-rt-24.

```
sftp> cd /usr/src/linux-2.6.29.6
sftp> put /ruta/del/parche.bz2
sftp> bye #Cierra la transferencia sftp # Ya se puede cerrar la transferencia
    sftp
```

A partir de aquí se puede trabajar únicamente con la terminal *shell SSH*. Para concluir se extrae el parche.

Código 6.11 Extraer el parche RT.

```
root@darkstar:~# cd /usr/src/linux-2.6.29.6
root@darkstar:/usr/src/linux-2.6.29.6# bzip2 -d patch-2.6.29.6-rt24.bz2
```

6.5 Parcheado y compilación del *kernel* 2.6.29.6-rt24 con capacidades RT

Una vez disponibles las fuentes y el parche RT en la placa, se procede a cambiar la versión por defecto del kernel 2.6.29.6 incluida con la distribución de Slackware 13.0 (ver el código 6.6 de la sección 6.3.2) para que soporte aplicaciones en tiempo real. Antes de compilar el kernel, y con el objetivo de habilitar esta nueva característica, es necesario añadir al código fuente una serie de líneas dispuestas a tal efecto. En este sentido, un parche es simplemente un diferecial de código que se incluye en la fuente y que permite personalizar el kernel, abriendo así una gama de aplicaciones más amplia que la disponible con el instalado por defecto en la distribución. Si bien es cierto que los kernel 2.6.xx.x incluyen soporte en tiempo real, éste es del llamado tipo *"soft"* (véase la sección 5) o sin prioridad absoluta de ejecución. Por todo ello, y para el propósito de un control estricto y preciso de las variables involucradas, se hace necesario aumentar el nivel de determinismo del SO para proporcionarle prioridad absoluta de ejecución (*Full Preempt*).

6.5.1 Parche 2.6.29.6-rt24

Como se ha comentado, el primer paso para incorporar estas nuevas funcionalidades es aplicar el correspondiente parche a las fuentes. A tal efecto, *Linux/Unix* dispone del comando *patch*. Téngase en cuenta que el directorio de trabajo debe ser aquél donde se encuentre extraído el parche, y que éste comando funcionará únicamente si se encuentra en la carpeta de las fuentes, lo cual justifica los directorios utilizados para transferir y extraer los archivos necesarios.

Código 6.12 Aplicar el parche.

```
root@darkstar:~# cd /usr/src/linux-2.6.29.6 # Directorio de las fuentes
root@darkstar:/usr/src/linux-2.6.29.6# patch p1 < patch-2.6.29.6-rt24
```

Tras ejecutar la orden aparecerán las operaciones que *patch* está realizando. El proceso terminará cuando se muestre por pantalla el prompt del *shell* (esto es, usuario@equipo:directorio/de/trabajo#). Con este sencillo comando se ha incluido la capacidad de tiempo real a las fuentes del *kernel*.

6.5.2 Compilación del *kernel 2.6.29.6-rt*

En esta sección se configurará el *kernel* parcheado para posteriormente compilarlo y conseguir que sea completamente funcional. Existe una gran cantidad de guías para compilarlo. En este trabajo se ha hecho uso del ejemplo publicado en la página oficial de la distribución de Slackware, disponibles tanto en inglés como en español. Se adjuntan los comandos empleados, así como comentarios explicativos sobre los mismos.

Código 6.13 Terminal 2: Opciones del del kernel.

```
root@darkstar:~# cd /usr/src/
root@darkstar:/usr/src# rm linux #Elimina el vínculo actual a la carpeta de las
     fuentes
root@darkstar:/usr/src# ln -s linux-2.6.29.6 linux #Crea un vínculo apuntando a
     ella
root@darkstar:/usr/src# zcat /proc/config.gz > /usr/src/linux/.config #Copia el
     archivo config del kernel actual a la carpeta de fuentes
root@darkstar:/usr/src# cd /usr/src/linux
root@darkstar:/usr/src/linux# make oldconfig #Configura el nuevo kernel con las
     opciones por defecto del antiguo. Al haberse añadido nuevas
    funcionalidades aparecerán también opciones adicionales.
```

```
root@darkstar:/usr/src/linux# make menuconfig #Configurador en entorno gráfico
```

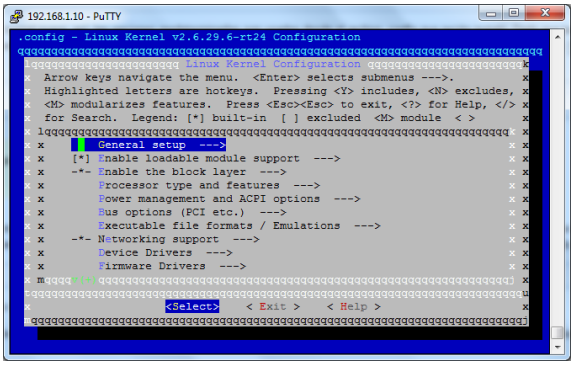


Figura 6.5 ventana principal de *menuconfig*.

El principal cambio que se debe realizar en *menuconfig* es seleccionar el modo de prioridad del sistema operativo. Existen cuatro tipos disponibles para el kernel 2.6.29.6 parcheado (El parche añade la última opción). También se debe añadir una terminación local al nombre del nuevo *kernel* en el campo *General Setup—>Local version - append to kernel release*, ya que se trata de la misma versión desde la que se está instalando. Si no se introdujera este campo, los módulos actuales se eliminarían. Aquí se ha optado por introducir la cadena de caracteres *pablo*.

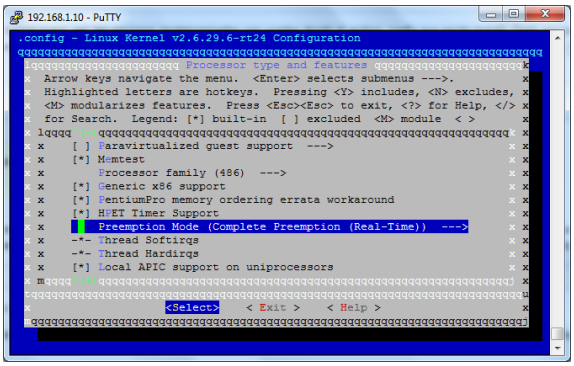


Figura 6.6 *Processor type and features–>Preemption Mode–>Complete Preemption (Real-Time).*

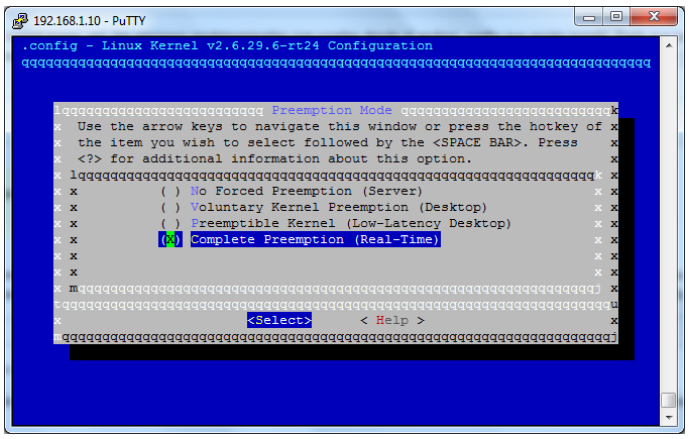


Figura 6.7 Tipos de prioridad.

Código 6.14 Compilando el kernel.

```
root@darkstar:/usr/src/linux# make bzImage modules# compila el kernel y los mó
    dulos
root@darkstar:/usr/src/linux# make modules\_install # instala los módulos en /
    lib/modules/<kernelversion>
root@darkstar:/usr/src/linux# cp arch/x86/boot/bzImage /boot/vmlinuz-custom
    -2.6.29.6 # copia el nuevo kernel
root@darkstar:/usr/src/linux# cp System.map /boot/System.map-custom-2.6.29.6 #
    copia System.map (opcional)
root@darkstar:/usr/src/linux# cp .config /boot/config-custom-2.6.29.6 # copia
    de respaldo de la configuración del kernel
root@darkstar:/usr/src/linux# cd /boot
root@darkstar:/boot# rm System.map # borra el vínculo antiguo
root@darkstar:/boot# ln -s System.map-custom-2.6.29.6 System.map # crea un
    nuevo vínculo
```

Configuración de *Lilo*

Como última tarea, será necesario modificar el archivo /etc/lilo.conf. *Lilo* es el acrónimo de *Linux Loader*, un programa de *Linux* que se ejecuta cuando se enciende la máquina y que permite elegir el SO de arranque. Toda la configuración referente al mismo se encuentra en el citado directorio. No se debe olvidar que tras modificar cualquier parámetro de tal fichero se deberá ejecutar */sbin/lilo* o simplemente *lilo* en el terminal.

La manera más simple de modificar el archivo es utilizar *Vim* (*Vi Improved*),del cual puede encontrarse una pequeña guía en el anexo C. Su contenido es el siguente: (Se omiten partes del código no relevantes para la explicación)

Código 6.15 lilo.conf.

```
# LILO configuration file
# generated by 'liloconfig'
#
# Start LILO global section
# Append any additional kernel parameters:
# Append=" vt.default_utf8=0 ide-core.nodma=0.0"
# serial = 0,9600n8
Append="console=tty0 console=ttyS0,115200n8 ide-core.nodma=0.0"
# Append="ide-core.nodma=0.0"
boot = /dev/hda
install=text

...

# Standard menu.
# Or, you can comment out the bitmap menu above and
# use a boot message with the standard menu:
message = /boot/boot_message.txt   # Mensaje de arranque que se mostrará por
    pantalla

# Wait until the timeout to boot (if commented out, boot the
# first entry immediately):
prompt
default = 3    # Arrancará por defecto la imagen con la etiqueta 3
# Timeout before the first entry boots.
# This is given in tenths of a second, so 600 for every minute:
timeout = 100
```

```
# Override dangerous defaults that rewrite the partition table:
change-rules
reset
# Normal VGA console
vga = normal

...

# End LILO global section

# Linux bootable partition config begins
image = /boot/vmlinuz
root = /dev/hda1
# Linux-xenomai
label = 1
read-only

image = /boot/vmlinuz-huge-2.6.29.6
root = /dev/hda1
# Linux-original
label = 2
read-only
image = /boot/vmlinuz-custom-2.6.29.6 #Nuevo bloque para incluir la imagen del
    kernel parcheado
root = /dev/hda1
# Linux-rt
label = 3
read-only

# Linux bootable partition config ends
```

Para incluir la nueva imagen de arranque en *lilo* basta con añadir un bloque de código bajo las imágenes ya disponibles con el mismo formato, editando pertinentemente los datos referentes a su nombre y etiqueta.

En este punto es importante hacer varias aclaraciones referentes a la modificación de este fichero, ya que influyen directamente sobre la capacidad de conexión a través de *SSH*. En relación a *lilo*, puede resultar ilustrativo conocer algunos aspectos básicos del proceso de arranque de *Linux*.

Cuando se arranca el sistema, al igual que en cualquier otro computador, la BIOS (*Basic Input/Output System*) es la encargada de ejecutar el *Master Boot Record* o MBR, un programa ubicado en el primer sector de la partición de arranque, que es el encargado de ejecutar el selector lilo, en el caso de una partición tipo Linux. A todos los efectos, el SO no se encuentra en ejecución, por lo que servicios como el servidor OpenSSH no estarán disponibles. Esto supone que la conexión ssh no sería posible, a menos que se configure lilo para que ejecute una de las opciones de arranque por defecto. Esto se puede realizar incluyendo una opción de arranque por defecto, en el campo *default = número de la etiqueta*. De esta manera lilo esperará el tiempo especificado en *timeout = tiempo de espera* y cargará la imagen marcada con tal etiqueta, iniciando así la inicialización de módulos entre los que se encuentra OpenSSH.

Si se desea también es posible puede cambiar el mensaje de arranque modificando el archivo */boot/boot_message.txt*. Téngase en cuenta, no obstante, que éste mensaje sólo será visible cuando se acceda al PC/104 a través de puerto serie, a la luz de lo explicado anteriormente.

Con objeto de comprobar si el proceso de compilación ha resultado satisfactorio, se reinicia la máquina con el comando *reboot*. Una vez concluido el arranque se ejecutan los comandos del código 6.6 para verificar la versión del kernel. El resultado por pantalla indica el éxito de la operación. Nótese la terminación personalizada de la versión del *kernel*, que incluye información referente al parche y el sufijo correspondiente a *Localversion* introducido por el usuario.

Código 6.16 Verificando los resultados.

```
root@darkstar:~# uname -r -v # Muestra la publicación (release) y la versión
2.6.29.6-rt24-pablo #2 PREEMPT RT
```

6.6 Banco de pruebas

Para rematar la sección y con el objetivo de asegurar la operación del sistema en tiempo real, se realizarán los pertinentes *tests* de rendimiento. Algunos de los más importantes son *cyclictest*, un programa sencillo que somete al procesador a carga desde el reposo y mide los tiempos de latencia y *hackbench*, que se ejecuta en paralelo al anterior y que simula un entorno de carga para el procesador en el caso de operaciones en tiempo real. Con estas dos herramientas se pueden conseguir datos fiables que caractericen el rendimiento de la máquina en el contexto de la computación en tiempo real.

6.6.1 Paquete *rt-tests-1.0*

Se trata de un paquete que incluye todas las herramientas necesarias para poner a prueba un sistema *RT*. Se pueden conseguir sus fuentes de la página *https://www.kernel.org/pub/linux/utils/rt-tests/*. Para compilarlas es necesario disponer de las librerías *NUMA (Non Uniform Memory Acess)* y algunos paquetes de desarrollador, que se descargan de *www.gnu.org/software/*. A continuación se facilita una lista de todas los paquetes y versiones que se han utilizado:

- *numactl 2.0.10*
- *Autoconf 2.65*
- *Automake 1.11.1*
- *libtool 2.2.10*
- *m4 1.4.14*

La manera de compilar cualquiera de ellos, salvo *numactl*, es transfiriendo los archivos comprimidos en extensión *tar.gz* mediante *sftp o psftp* a cualquier directorio de *slackware*. Aquí se ha elegido */usr/src*. Se extraen en la misma carpeta, se cambia el directorio de trabajo a la ruta recién creada con *cd* y se visualiza el archivo *INSTALL* con la orden cat INSTALL. Todas las instrucciones que hacen referencia al paquete se encuentran recogidas en él.

Para el caso de *numactl* el procedimiento es distinto. De acuerdo con el archivo *INSTALL* de las fuentes, se debe ejecutar en primer lugar *./autogen.sh*, para lo cual es necesario haber instalado previamente los paquetes de desarrollador anteriormente descritos en las versiones adecuadas. Si el sistema no dispone de ellas devolverá una descripción detallada por pantalla del error donde se especifica la librería o versión requeridas.

A la hora de instalar el paquete principal *rt-tests-1.0* se obtuvo un error de compilación, que hacía referencia a la no definición de las *scheduling policies*, variables utilizadas en el código de las pruebas. En [18] se encuentra la solución a este problema. Tal y como indica la tipología del error, es necesario transferir el archivo *pidstat.h* con estas variables definidas a */usr/src/rt-tests-1.0/src/include* y modificar con *Vim* el archivo *rt-utils.c* de */usr/src/rt-tests-1.0/src/lib* añadiendo la línea *include <pidstat.h>* en la cabecera. Una vez guardados los cambios se ejecutó *make* y la compilación se llevó a cabo sin problemas. Para otros tipos de error se puede consultar [19].

6.6.2 *Cyclictest y Hackbench*

Como ya se ha comentado al principio de la sección 6.6.1, se trata de las dos herramientas principales de monitorización de las capacidades en tiempo real de un sistema operativo. Las principales opciones con las que se pueden ejecutar se encuentran detalladas en */usr/src/rt-tests-1.0/README.markdown*.

Entre las consideraciones recogidas en este manual se exponen las principales características recomendadas de un test de latencia o capacidad *RT*. A partir de ellas se decide crear un *test* con las siguientes:

Tabla 6.1 Diseño de test para el *kernel*.

Característica	Valor
Latencia máxima admisible	500ms
Tiempo de ejecución	24 h
Número de hilos	1
Tiempo entre hilos	1000μs
Registro de eventos	*/sys/kernel/debug/tracing/trace*

Además, mientras se lleva a cabo la evaluación, *hackbench* se encuentra sometiendo al sistema a carga transmitiendo mensajes de 1Mb. Para hacer que la ejecución de este programa tenga una mayor duración, se fija el número de bucles en 1×10^6. Queda por tanto esperar a que se realice el test y extraer las conclusiones, comentadas a continuación.

6.6.3 Resultados

Se exponen a continuación los resultados del banco de pruebas del *kernel 2.6.29.6-rt24* descrito en la sección 6.6.2.

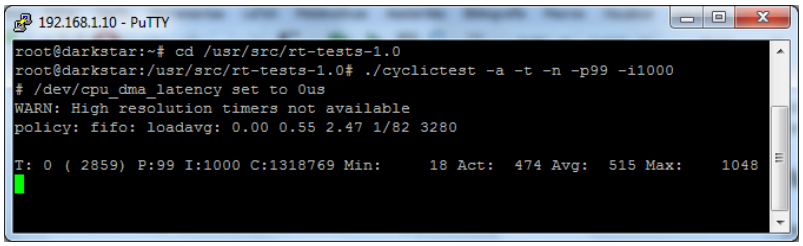


Figura 6.8 Resultados para el *kernel 2.6.29.6-rt24-pablo*.

Como puede observarse en la figura 6.8, la última columna indica la latencia máxima obtenida durante la prueba, que es de 1048 ms. Este resultado es aceptable en primeros términos para el *hardware* y las versiones tanto de *slackware* como del *kernel* utilizadas. Nótese también la media de 515ms en el campo *Avg*.

Una manera de cuantificar la mejora respecto al *kernel 2.6.29.6* sin parchear es sometiéndolo a ensayo y mostrando los valores máximos de latencia obtenidos, para lo cual se debe iniciar *slackware* con dicha versión, haciéndose necesario modificar el archivo *lilo.conf* tal y como se explicó en la sección 6.5.2. Véase cómo en este caso la latencia más desfavorable es bastante superior que para el caso del *kernel* tipo *PREEMPT RT*.

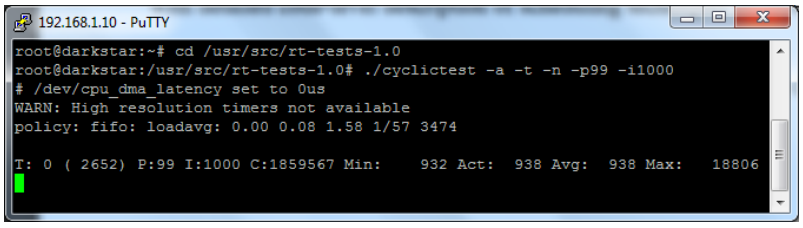


Figura 6.9 Resultado para la versión *2.6.29.6-generic*.

A modo ilustrativo, se ha realizado la misma prueba en un procesador de ordenador portátil *Intel Pentium Dual Core T3200* de dos núcleos a 2GHz con la distribución *Ubuntu 16.10* y el kernel por defecto *4.8.0-59-generic* sin funcionalidades en tiempo real. Los resultados de la comparación se muestran a continuación en la figura 6.10

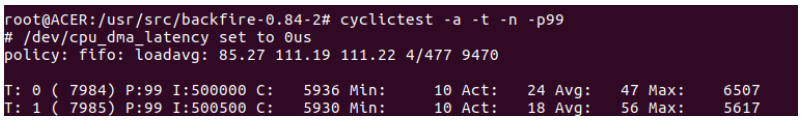


Figura 6.10 Prueba en *Intel Pentium Dual Core*.

Al disponerse de dos núcleos, para comprobar la latencia *Cyclictest* envía un hilo con prioridad 99 a cada uno. Obsérvense los valores máximos del *Dual Core*, hasta seis veces por encima del *Vortex*.

7 Conclusiones y líneas futuras de trabajo

Para finalizar se presentan las conclusiones obtenidas de los estudios llevados a cabo en las secciones precedentes.

7.1 Viabilidad de *Linux-RT*

Con base en la cifras presentadas en la sección 6.6.3es posible afirmar que el sistema operativo *GNU/Linux* basado en la distribución *slackware* con *kernel* personalizado *2.6.29.6-rt24* es teóricamente capaz de soportar aplicaciones en tiempo real, lo cual implica que trabajaría con latencias lo suficientemente bajas como para no comprometer el control del vehículo ante cualquier eventualidad de *software* referente al sistema operativo, a expensas de ser implementado en la ECU instalada en el FOX, llevando a cabo la congruente programación de la misma y realizando pruebas de campo en las condiciones habituales estipuladas por el FCCL o el organismo competente.

De esta manera, en la sección 6 se ha presentado una guía completa de personalización del *kernel* de *Linux* en el ámbito del control en tiempo real, con todas las fuentes necesarias y con la resolución de algunos de los problemas más comunes que pueden surgir durante el proceso, que debería servir de referencia para futuras operaciones tanto a la hora de cambiar o actualizar la distribución de Linux o del *kernel*.

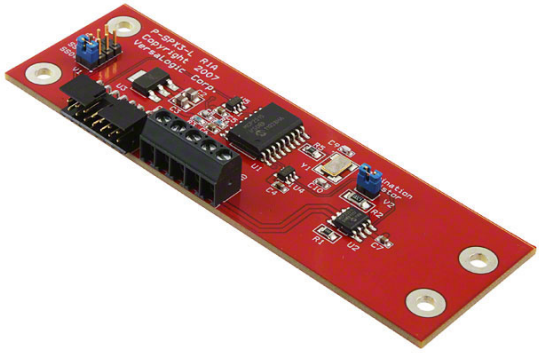
Todo lo comentado a propósito de la implantación del *kernel* de *Linux* posee evidentemente aplicación en cualquier campo de la electrónica embarcada, desde UAV's hasta maquinaria industrial automatizada, domótica, armamento,...etc. Es esta amplia variedad de utilidades la que hace valer la pena el conocimiento adquirido durante la realización de este estudio.

7.2 Propuesta de mejora de la ECU

A lo largo de los dos primeros bloques del presente trabajo se estudiaron y describieron con detalle el sistema de partida y algunas de las principales alternativas que ofrece el mercado actual para conseguir una configuración moderna y compacta en lo referente a la unidad de control electrónico de un vehículo embarcado como el FOX. A la vista de este estudio, se proponen dos nuevas configuraciones que se detallan a continuación.

7.2.1 *SBC Copperhead*

En virtud de las características detallas en la sección 4.2.3, este SBC de *Versalogic inc.* compone la mejor de las alternativas de control para un vehículo como el FOX principalmente por los potentes procesadores que ofrece y la capacidad de expansión y miniaturización que permite el factor de forma *EBX*. Para adaptarlo al entorno de control del FOX se deberían utilizar módulos PC/104 que implementaran controladores de bus CAN, bien sean los propuestos por esta compañía (ver figura 7.1a) u otros con factor de forma PC/104 (7.1b)



(a) Tarjeta *VL-SPX-3* de *Versalogic*([20]).

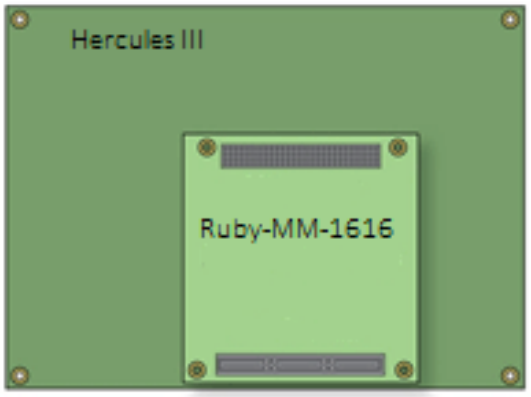
(b) Módulo CAN PC/104 de *Diamond Systems*.

Figura 7.1 Tarjetas CAN para expansión de la placa *Copperhead*.

7.2.2 *Hércules III*

La configuración que ofrece un mayor número de posibilidades a la vista de las necesidades de control analizadas consistiría en la implantación a largo plazo del computador *SBC Hercules III* descrito en la sección 4.2.1 conjuntamente con el módulo de adquisición de datos *RUBY-MM-1616*, versión actualizada de la actualmente implantada en el vehículo 1612.

(a) Tarjeta *Ruby-MM-1616AP de Diamond Systems [21]* .



(b) Configuración de la nueva ECU.

Figura 7.2 Segunda propuesta.

De esta manera se dispondría de un equipo preparado para afrontar los horizontes futuros del proyecto FOX, tanto en *hardware* como en *software*. No obstante es perfectamente posible continuar utilizando el *hardware* y *software* actuales, hecho que no perjudicará al avance del proyecto en los plazos corto y medio.

7.3 Trabajo pendiente

Para establecer un plan de acción a seguir hasta la completa instauración de la nueva ECU, se proponen varias tareas:

- Adquirir una licencia de la versión más actual del SO *QNX* y evaluar su comportamiento en el vehículo.
- Reescribir el código de sus controladores, actualmente programados en lenguaje *QNX*, para que corran en *Linux*. Esta tarea podría constituir en sí misma una motivación para la realización de nuevos proyectos bajo la dirección del FCCL por los alumnos de ingeniería industrial o de telecomunicaciones con mayor conocimiento en la materia.
- Considerar la posibilidad de operar la ECU con un SO dual *QNX+Linux RT* para estudiar el comportamiento de *Linux* directamente en el vehículo en lugar de en banco de ensayos. Esto es posible creando particiones del disco dedicadas tanto a la instalación de *QNX* como de *Linux* parcheado en tiempo real, y utilizando un menú de selección que se muestre por pantalla al arranque de la máquina para seleccionar la deseada. Aunque flexible a la hora de estudiar el comportamiento de ambos OS, esta opción supondría una reducción de memoria disponible para cada uno por separado.

Además,como trabajo adicional en el ámbito de la prueba con el RTOS basado en *Linux* se proponen estudios para minimizar la latencia monitorizada. Para ello se debería hacer uso de la herramienta *ftrace* del *kernel*, que se encarga de registrar los procesos con mayor latencia y que se encuentra ubicada en el directorio mencionado en la tabla 6.1. La manera de acceder a ella es recompilar el *kernel* habilitando las opciones de *menuconfig* que hacen referencia al registro de eventos, recogidas en la pestaña *kernel tracing* (figura 7.3). También cabe la posibilidad de añadir al montaje del laboratorio una tarjeta o módulo de controlador CAN 2.0 con factor de forma PC/104 y elaborar la documentación necesaria que permita al personal del mismo realizar ensayos con dicho componente.

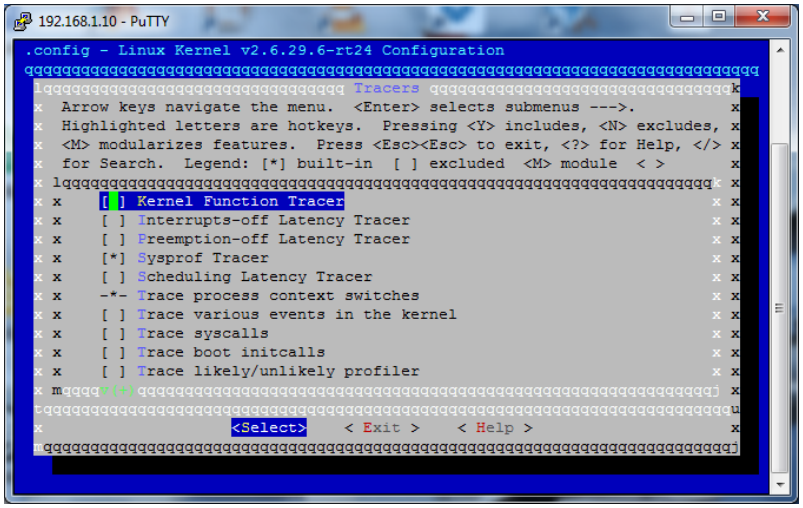


Figura 7.3 Habilitar los *tracers* en *menuconfig*.

Apéndice A
Controller Area Network (CAN) [1]

A.1 Introducción

El bus CAN fue desarrollado por *BOSCH* como un sistema de transmisión de mensajes multi-maestro que especifica una velocidad máxima de transferencia de 1 megabit por segundo (Mbps). A diferencia de otras redes tradicionales como USB o Ethernet, el CAN no envía grandes bloques de información punto a punto desde un nodo A hasta uno B bajo la supervisión de un bus central maestro. En una red CAN se emite un gran número de mensajes cortos a totalidad de la red, lo cual favorece la consistencia de la información en cada nodo del sistema.

A.2 El estándar CAN

El *Controller Area Network* es un bus de comunicación en serie definido por la Organización Internacional de la Estandarización (ISO), originalmente concebido para reemplazar el complejo sistema de cableado por un bus de dos conectores en la industria del automóvil. La especificación impone alta inmunidad a la interferencia eléctrica y la habilidad de autodiagnosticar y reparar los errores en los datos manejados. Estas características han propiciado la popularidad del CAN en una amplia variedad de industrias, incluyendo la construcción, automatización, medicina y fabricación.

El protocolo de comunicaciones CAN, *ISO-11898:2003*, describe cómo se transmite la información entre dispositivos de una red y cumple con el modelo de Interconexión de Sistemas abiertos (OSI) que se define en términos de capas. La comunicación real entre dispositivos conectados al medio físico se define por las capas físicas del modelo. La arquitectura de la *ISO 11891* define las dos capas más bajas de las 7 presentes en el modelo OSI/ISO, así como la capa de enlace de datos (figura A.1)

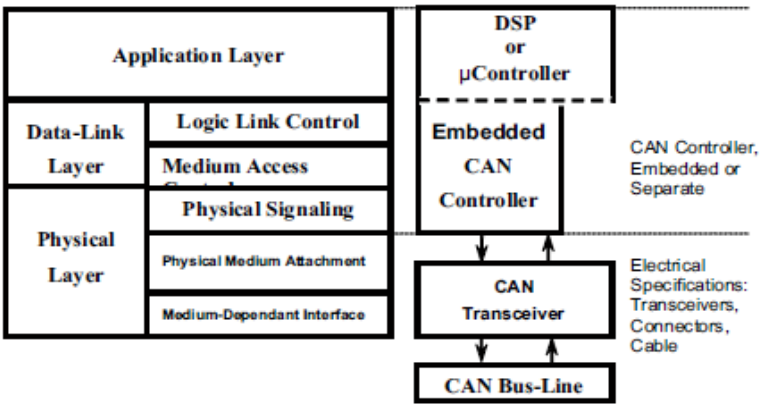


Figura A.1 Arquitectura ISO 11898.

A.3 CAN Estándar y extendido

El protocolo de comunicación CAN es de tipo *CSMA/CD+AMP*. CSMA implica que cada nodo del bus debe esperar un periodo predeterminado de inactividad antes de intentar enviar un mensaje. CD+AMP significa que las colisiones se resuelven a través de un arbitraje a nivel de bit, basado en una prioridad preprogramada de cada mensaje en el campo del identificador. El identificador con prioridad más alta siempre gana el acceso al bus. Esto es, el último pulso lógico alto continúa transmitiendo porque tiene la prioridad más alta. Como cada nodo de un bus toma parte en escribir cada bit como si se estuviera escribiendo, un nodo cualquiera conoce si situó el bit lógico alto en el bus.

El estándar *ISO-11898:2003*, con el identificador de 11 bits, proporciona velocidades de transmisión de datos desde 125 kbps hasta 1 Mbps. Al estándar se añadió posteriormente el identificador extendido de 29 bits. El primero es capaz de proporcionar 2048 identificadores distintos mientras que el de 29 bits llega hasta los $2^{29} = 537$ millones.

A.3.1 Trama de bits en CAN estándar y extendido

Estándar

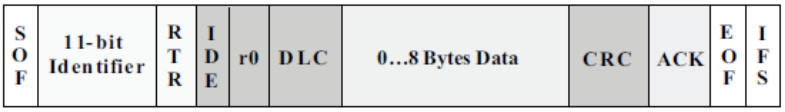


Figura A.2 Trama de bits estándar.

SOF La entrada individual dominante del mensaje (SOF por sus siglas en inglés *Start Of Frame*) marca el principio de la trama, y se utiliza sincronizar los nodos tras encontrarse en un estado de inactividad.

Identificador El identificador CAN estándar de 11 bits establece la prioridad del mensaje. Cuanto más bajo el valor binario, más alta será la prioridad.

RTR Petición de transmisión remota, consiste en un bit que es dominante cuando la información es requerida por otro nodo. Todos los nodos reciben la petición, pero el identificador determina el nodo especificado. La información de respuesta también es recibida por todos los nodos y utilizada por cualquier nodo interesado. De esta manera se hace un uso uniforme de todos los datos que circulan por el sistema.

IDE-A Un identificador individual dominante de extensión (IDE) implica que se está transmitiendo un identificador CAN estándar sin extensión.

r0 Bit reservado para una posible ampliación futura.

DLC El código de longitud de datos (DLC) contiene el número de bits datos que se están transmitiendo.

CRC Comprobador de redundancia cíclica de 16 bits, contiene el número de bits de la datos de aplicación anterior referentes a detección de errores.

ACK Cada nodo que reciba un mensaje preciso sobreescribe este bit recesivo en el mensaje original con uno dominante, indicando que se ha enviado un mensaje libre de errores. Cuando un nodo receptor detecta un error deja este bit recesivo y descarta el mensaje, y el nodo transmisor lo vuelve a enviar tras un nuevo arbitraje. De esta manera, cada nodo reconoce la integridad de su información. La longitud de la región ACK es de dos bits, uno es el de reconocimiento y otro el delimitador.

EOF Fin de la trama (*End Of Frame*) es un campo de 7 bits que marca el final de un mensaje CAN y desactiva el relleno de bits, indicando un error de relleno cuando es dominante. Cuando 5 bits del mismo nivel lógico se suceden durante la operación normal la información se rellena con un bit de nivel lógico opuesto.

IFS Espacio entre tramas con una longitud de 7 bits. Contiene el tiempo requerido por el controlador para mover una trama correctamente recibida a su posición correcta en el área de buffer del mensaje.

Extendida

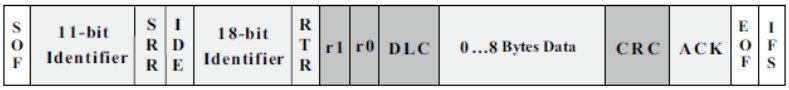


Figura A.3 Trama de bts extendida.

Como es posible notar en la figura A.3, el mensaje CAN extendido tiene el mismo formato de trama que el estándar, con la adición de los siguientes campos:

SRR El bit de petición remota de sustitución SRR reemplaza al bit RTR en la posición en la trama estándar como marcador de lugar.

IDE Un bit recesivo en la extensión del identificador indica que existen más bits de este campo a continuación. La extensión de 18 bits va luego de este IDE.

r1 En sucesión a los bits RTR y r0, se incluye un bit adicional de reserva delante del bit DLC.

A.4 Mensaje CAN

A.4.1 Arbitraje

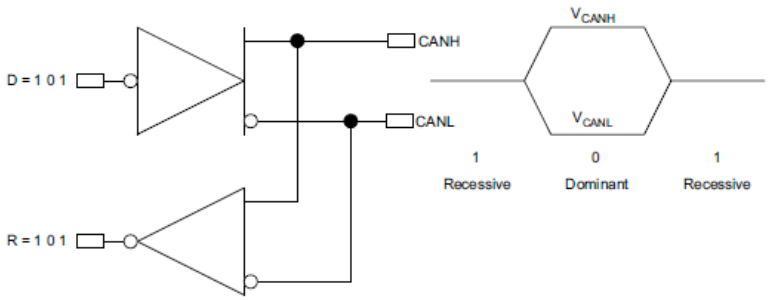


Figura A.4 Lógica del CAN.

En la figura A.4 se puede observar una característica fundamental del bus CAN: El bus, y las entradas del transmisor y las salidas del receptor se encuentran en estados lógicos opuestos. Normalmente, un valor lógico elevado se asocia a un 1 y a uno bajo un cero, pero en un bus CAN los valores son opuestos.

El acceso al bus está activado por eventos, y tiene lugar de forma aleatoria. Si dos nodos tratan de acceder al bus simultáneamente, se implementa el acceso mediante un arbitraje no destructivo y bit a bit. No destructivo se refiere a que el nodo que gana el arbitraje continua con el mensaje, sin que éste sea destruido o corrompido por ningún otro nodo.

La inclusión de prioridad en los mensajes en el identificador es una característica del bus CAN que lo hace particularmente atractivo para su uso en entornos de control en tiempo real. Cuanto más pequeño sea el número del identificador, mayor será la prioridad. Un identificador formado por sólo ceros es el mensaje de mayor prioridad en una red porque es el que mantiene al bus dominante durante más tiempo. Por tanto, si dos nodos comienzan a transmitir simultáneamente, el nodo que envía el último bit del identificador cero (dominante) retendrá el control del Bus frente a otro con bit final del identificador 1 (recesivo). Un bit dominante siempre sobreescribe a uno recesivo en un bus CAN.

Nótese que un nodo transmisor monitoriza constantemente cada bit de su propia transmisión. Este es el motivo por el cual el transceptor de la figura A.4 en la cual las salidas CANH y CANL del transmisor están conectadas internamente a la entrada del receptor. El retraso de propagación de una señal en el bucle interno desde la entrada del transmisor hasta la salida del receptor se usa típicamente como una medida cualitativa que define a un transceptor CAN, y es conocida como tiempo de bucle (*loop time*).

La figura A.5 muestra el proceso de arbitraje manejado automáticamente por un controlador CAN. Como cada nodo monitoriza sus propias transmisiones continuamente, tan pronto como el bit recesivo del nodo B se reescribe por el bit de mayor prioridad dominante de C, B detecta que el estado del bus no coincide con el bit que transmitió. Por tanto, el nodo B para la transmisión mientras que el nodo C continua transmitiendo su mensaje. El nodo B realizará otro intento de transmisión en cuanto el C deje libre el bus. Esta funcionalidad es parte de la capa de señalización física establecida por la norma ISO 11898, lo cual significa que se encuentra contenida por completo en el controlador CAN y es completamente transparente de cara al usuario.

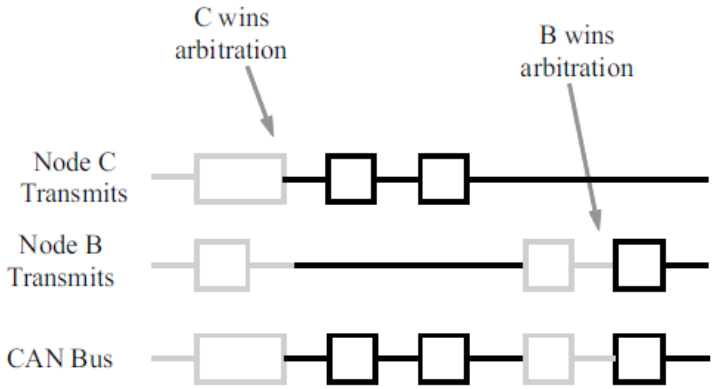


Figura A.5 Arbitraje de un bus CAN.

A.4.2 Tipos de mensaje

Los cuatro tipos distintos de mensaje o tramas que se pueden transmitir a través de un bus CAN son de datos, remotos, de errores y de sobrecarga.

Mensaje de datos Es el tipo más común, y comprende los campos de Arbitraje, de Datos, CRC y ACK. El campo de arbitraje contiene un identificador de 11 bits (figura A.2) y el bit RTR, que es dominante para tramas de datos. En la figura A.3 se puede observar el identificador de 29 bits, y el bit RTR. El siguiente es el campo de Datos, que contiene entre cero y ocho bytes de datos, y el campo CRC que contiene la comprobación de 16 bits para detectar errores. Por último se encuentra el campo ACK.

Mensaje remoto El propósito de esta trama es solicitar la transmisión de datos desde otro nodo. El mensaje Remoto es similar al de Datos, con dos diferencias importantes. Primero, este tipo de mensaje se marca explícitamente como Remoto mediante un bit recesivo RTR en el campo de arbitraje, y segundo no hay datos.

Mensaje de error El mensaje de error es una transmisión especial que viola las reglas de formato de un mensaje CAN. Se transmite cuando un nodo detecta algún error en un mensaje, y provoca que otros nodos envíen también una transmisión de error. El transmisor original retransmite entonces el mensaje automáticamente. Un elaborado sistema de contadores de error en el controlador CAN se asegura de que un nodo no pueda colapsar un bus transmitiendo mensajes de error repetidamente.

Mensaje de sobrecarga Es similar al de error en cuanto a formato, y lo transmite un nodo que se encuentra demasiado ocupado. Se utiliza fundamentalmente para evitar retrasos entre mensajes.

Apéndice B
Distribución *Slackware*

Figura B.1 Logotipo de la distribución *Slackware*.

En este apéndice se recogen los pasos necesarios para instalar la distribución de *GNU/Linux Slackware* en una tarjeta *SSD* de tipo *Compact Flash*. Si bien para la prueba que se ha realizado en este trabajo se ha partido de la tarjeta con *Slackware* 13.0.0.0 convenientemente instalado, resulta práctico conocer el proceso para futuras referencias, debido a que se trata de una distribución que ofrece diversas opciones en este aspecto.

La distribución oficial de Slackware fue desarrollada por Patrick Volkerding con el objetivo de ser sencilla y estable. Cumple con todos los estándares publicados de *Linux*, tal y como el relativo al sistema de archivos.

Se encuentra disponible tanto en 32 como en 64 bits, y las versiones actuales se encuentran basadas en la versión 4.4 del núcleo o *kernel Linux* y en la versión 2.23 de la librería C de GNU. Contiene un programa de instalación fácil de usar, documentación *on-line* extensiva y un sistema de paquetes basado en menús. Puede funcionar desde en procesadores *Pentium* como en máquinas más actuales y sofisticadas de tipo x86 y x86_64.

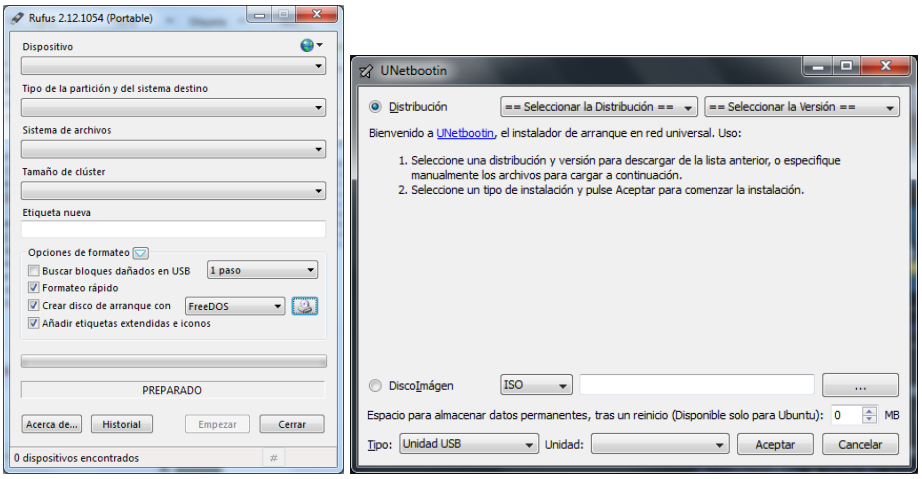
B.1 *Compact Flash Bootable*

Un medio extraible que contiene un SO instalado y que puede utilizarse como disco de arranque del sistema recibe el adjetivo anglosajón de *bootable*. Para dotar a una tarjeta *Compact Flash* de esta funcionalidad (hacerla *bootable*) son necesarias dos herramientas, además de la imagen *ISO* de la distribución que se desee instalar (En este caso la de *Slackware*, que se puede conseguir desde la página web *http://www.slackware.com/*).

La primera de ellas es un lector de tarjetas con un terminal *USB*, con objeto de conectar la *Compact Flash* al equipo desde el que se quiera realizar el proceso. La segunda herramienta consiste en un programa que escriba el medio extraible con el formato de arranque adecuado. Existe una gran variedad de software gratuito en la red destinado a este cometido, pero basta con nombrar *UNetbootin* (*https://unetbootin.github.io/*) y *Rufus* (*https://rufus.akeo.ie/?locale*).

Una vez se dispone de todo lo anterior se procede como sigue:

- Se abre el programa *UNetbootin* o *Rufus*. Las ventanas principales de cada uno se presentan en las figuras B.2a y B.2b.



(a) *Rufus* . **(b)** *UNetbootin* .

Figura B.2 Ventanas principales de los dos programas elegidos para crear una *Compact Flash Bootable*.

- En el caso de *Rufus*, se selecciona el dispositivo extraible en la primera pestaña y la imagen *ISO* de la distribución seleccionando esta opción junto a *"crear disco de arranque con"*. Por último se hace *click* en *"Empezar"*. *Para UNetbootin* Se selecciona *"DiscoImagen"* y se busca la ubicación de la imagen *ISO*. Como últimos pasos se selecciona la unidad *USB* donde se encuentra insertado el lector de tarjetas y se hace *click* en *"Aceptar"*

Una vez terminado el proceso por uno u otro programa se estará en disposición de una tarjeta *Compact Flash* desde la que se puede arrancar cualquier computador si se selecciona la opción de adecuada a cada programa administrador de la BIOS.

B.2 Instalación

Para visualizar el proceso de instalación en esta prueba, tal y como se explicó en la sección 6.2.2 será necesario conectarse al dispositivo PC/104 a través de puerto serie. Una vez encendido el computador y si el proceso explicado en la anterior sección se ha completado de modo satisfactorio, comenzará el proceso guiado de instalación de *Slackware*. En esta sección se comentan los aspectos más relevantes en dicho aspecto.

B.2.1 Comando *cfdisk*

Permite crear particiones en el dispositivo de memoria principal en un entorno de menús. La primera vez que se abre muestra el espacio disponible en */dev/hda*, ver figura B.3.

Figura B.3 Menú de *cfdisk*.

Para poder instalar *Slackware*, tal y como se recomienda en la guía de instalación integrada, será necesario disponer de dos tipos de particiones:

- Una de tipo lógico con sistema de archivos *Linux Swap*. Se utiliza como extensión de memoria RAM cuando ésta se encuentra al límite de su capacidad. El espacio en disco que se debe asignar a esta partición debería encontrarse en el orden de un cuarto del tamaño del disco.
- Una primaria con sistema de archivos *Linux ext3*, marcada como *bootable* desde el menú principal. Puede utilizarse la memoria restante tras crear la partición *swap* para crear una sola de tipo primaria o crear hasta cuatro, marcando siempre una de ellas como *bootable*. Más información en

Tras crearlas, la lista debería figurar como se indica en la figura B.4

Figura B.4 Particiones necesarias del disco.

Una vez realizadas las particiones se procede a ejecutar el comando *setup* en el *shell*. Casi todas las opciones se pueden dejar por defecto, aunque se debe seleccionar la extensión de archivos *ext3*

Apéndice C
Comandos de *GNU/Linux*

En este apéndice se adjuntarán los principales comandos útiles para moverse por el entorno de *GNU/Linux* que se han utilizado. La sintaxis básica es como sigue: *comando -modificador*, indicándose en cualquier otro caso la correspondiente a la orden concreta. Es preciso señalar que los comandos más específicos como *patch* y *make* se encuentran debidamente explicados en la sección 6.5.1

La intención de la tabla C.1 es la disponer de una referencia rápida de las órdenes, si bien es posible disponer de información más detallada sobre las mismas y sus modificadores sin más que teclearlas seguidas de un espacio y - *-help* en el *shell* de *Linux*.

C.1 Comandos *shell*

El *shell* es la interfaz que existe entre usuario y dispositivo. Los principales tipos incluyen los *shell* de comandos y gráficos, siendo éstos últimos los que conforman la capa de más alto nivel en los SO comerciales en la actualidad. La apariencia de un *shell* de comandos, utilizado en la realización de este trabajo, consiste en una pantalla con fondo de color negro en la que aparece una cadena de caracteres conocida como *shell prompt*, compuesta por *nobre de usuario @ nombre del servidor* y seguida del directorio de trabajo. El programa que traduce las órdenes y el lenguaje en que se escriben se denomina *BASH*, siglas de *Bourne Again SHell*.

Tabla C.1 Tabla de comandos *shell*.

Comando	Descripción
clear	Limpia la pantalla del *shell*.
ls	Muestra una lista de archivos del directorio actual. Si se quieren visualizar también los archivos ocultos (precedidos por un punto) se deberá utilizar el modificador *-a*.
cd	Cambia el directorio al especificado (se escribe dejando un espacio).
cp	Copia de archivos. La sintaxis de esta orden es *cp /ruta/del/archivo/a/copiar /ruta/de/destino*.
halt	Apaga el sistema operativo.
ssh	Establece la conexión *SSH*. Para ello es necesario escribir junto al comando *usuario@IP*.
rm	Elimina archivos. Se puede forzar la operación con el modificador *-rf*.
sftp	*Secure File Transfer Protocol*. Sirve para transferir archivos entre terminales conectados por *SSH*. La sintaxis es la misma que en el comando *ssh*.
uname	Proporciona información sobre el kernel y su versión. Puede restringirse la información mostrada (sólo la versión) con el modificador -r
free	Uso de memoria.
ifconfig	Recoge la información de las redes cableadas e inalámbricas.

Figura C.1 Logotipo de *Vim*.

C.2 Comandos Vim

Vim (*Vi Improved*) es un editor de texto que se encuentra instalado por defecto en la mayoría de las distribuciones actuales de *GNU/Linux*. Aunque se trata de una herramienta muy potente, las órdenes bajo las que trabaja resultan radicalmente distintas a las conocidas por un usuario medio en cuanto a editores de texto.

Para editar cualquier documento de texto se escribe en el *shell* el comando *vim* seguido de la ruta del archivo. Aparecerá entonces su contenido por pantalla y se podrá modificar por conveniencia. Si el nombre del archivo no existe en el directorio especificado, se creará uno nuevo con el nombre indicado.

Vim posee dos entornos o modos principales claramente diferenciados:

- El modo *Normal* permite introducir los comandos con la sintaxis de *Vim*, como búsqueda de palabras en el texto, escritura en disco de las modificaciones realizadas, salir del editor... Además permite mover el cursor a través del texto y seleccionar partes del mismo con el llamado *modo visual*. Es el modo en el que el usuario se encuentra por defecto al acceder a un archivo. Puede presionarse la tecla *esc* para asegurar que se encuentra en el modo *Normal*
- Modo *Insertar*, con el que es posible modificar el texto. Se accede tecleando *i* en el modo *Normal*.

Las principales acciones desde el modo *Normal* se recogen en la siguiente tabla.

Tabla C.2 Principales comandos de *Vim*.

Comando	Descripción
i	Entra en el modo *Insertar*.
h	Mueve el cursor hacia la izquierda.
j	Mueve el cursor hacia abajo.
k	Mueve el cursor hacia arriba.
l	Mueve el cursor hacia la derecha.
:q	Cierra *Vim* sin guardar cambios.
:wq	Salir guardando cambios.
x	Elimina los caracteres sobre los que se sitúa.
v	Modo visual.
o	Abre una línea de texto nueva y entra en el modo *Insertar*.

Índice de Figuras

Índice de Tablas

Bibliografía

[1] Steve Corrigan and Industrial Interface. Introduction to the Controller Area Network (CAN). *Texas Instruments*, (August 2002):1–17, 2016.

[2] Isabelle Ferain, Cynthia A. Colinge, and Jean-Pierre Colinge. Multigate transistors as the future of classical metal–oxide–semiconductor field-effect transistors. *Nature*, 479(7373):310–316, 2011.

[3] David Miles and Andrew Scott. Technological Progress. *Macroeconomics: Understanding the Wealth of Nations*, pages 116–138, 2002.

[4] D Marcos. *Contributions to Power Management and Dynamics Control in Hybrid Vehicles*. PhD thesis, Escuela Técnica Superior de Ingeniería, Universidad de Sevilla, 2014.

[5] Diamond Systems. The benefits of PC/104. *http://www.diamondsystems.com/pc104*.

[6] Advantech. Pcm-3370 datasheet. *http://download.advantech.com/ProductFile/Downloadfile4/1-H4CC1/PCM-3370_Datasheet_ed.1.pdf*.

[7] Advantech. Pcm-3718-ho datasheet. *http://advdownload.advantech.com/productfile/PIS/PCM-3718HG/Product%20-%20Datasheet/PCM-3718HG_DS20140508140600.pdf*.

[8] Diamond Systems inc. Ruby-mm-1612 datasheet. *http://www.diamondsystems.com/files/binaries/rmm412datasheet.pdf*.

[9] IEI Technology Group inc. Pm-p005 datasheet. *http://www.attro.com/download/datasheet/PM-P005.pdf*.

[10] Electropaedia. Battery management system. *http://www.mpoweruk.com/bms.htm*.

[11] Micro/Sys. Sbc2596 datasheet. *http://www.embeddedsys.com/subpages/products/images/pdf/microsys_sbc2596_datasheet.pdf*.

[12] rock-robotics.org. Effect of i/o and operating system scheduling. *http://rock-robotics.org/stable/documentation/data_processing/timestamping.html*.

[13] Thomas W. Burger. Embedded development - qnx or linux? *https://software.intel.com/en-us/articles/embedded-development-qnx-or-linux*.

[14] Tri-M Technologies inc. Vdx104 datasheet. *https://www.tri-m.com/products/trim/files/specs/vdx104_spec.pdf*.

[15] Tri-M Technologies inc. Pm-005 datasheet. *https://www.tri-m.com/images/HE104_Spec_Sheet.pdf*.

[16] kernel.org. Fuentes del kernel linux 2.6.29.6. *https://www.kernel.org/pub/linux/kernel/v2.6/*.

[17] kernel.org. Parche rt del kernel linux 2.6.29.6. *https://www.kernel.org/pub/linux/kernel/projects/rt/*.

[18] Sebastien Godard. pidstat.h. *https://github.com/sysstat/sysstat/commit/40046f9846ece5f01cf53451e95d0f7f84d53989#diff-6a7ce30c53982c3172b6236cc7411bb0*.

[19] sh autogen.sh fails with error - possibly undefined macro: Ac_search_libs. *https://github.com/tmux/tmux/issues/257*.

[20] Digi-key electronics. Vl-spx-3. *https://www.digikey.com/product-detail/es/versalogic-corporation/VL-SPX-3/1241-1065-ND/3585760*.

[21] Diamond Systems. Ruby-mm-1616ap. *http://www.diamondsystems.com/products/rubymm1616ap*.